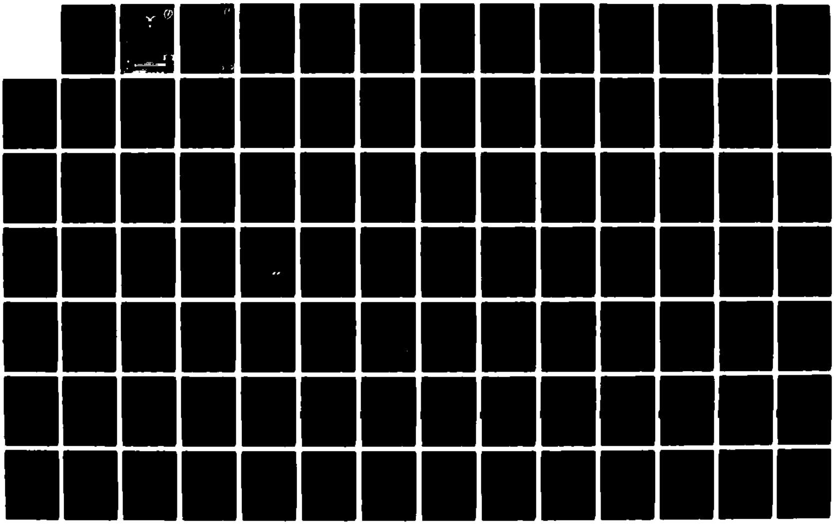
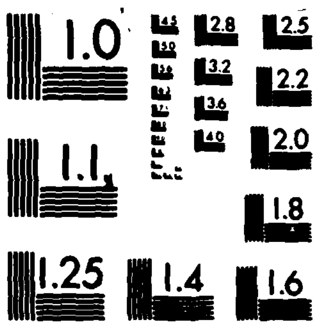
AD-A138 465 EFFECTS OF COMPUTER ARCHITECTURE ON FFT (FAST FOURIER TRANSFORM) ALGORITH..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M A MEHALIC 1/3

UNCLASSIFIED DEC 83 AFIT/GE/EE/83D-47 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFIT/GE/EE/83D-47

Accession For		
NTIS GRA&I	X	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution/		
Availability Codes		
Dist	Avail and/or Special	
A/1		

DTIC COPY INSPECTED 2

EFFECTS OF COMPUTER ARCHITECTURE

ON FFT ALGORITHM PERFORMANCE

THESIS

AFIT/GE/EE/83D-47

Mark A. Mehalic
1st Lt USAF

Approved for public release; distribution unlimited

AFIT/GE/EE/83D-47

EFFECTS OF COMPUTER ARCHITECTURE ON FFT ALGORITHM PERFORMANCE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

by

Mark A. Mehalic, B.S.E.E.
First Lieutenant, USAF

December 1983

Approved for public release; distribution unlimited.

Acknowledgements

I would like to thank my thesis advisor, Dr. Pedro Rustan, for proposing this topic, and for providing the guidance, enthusiasm, and confidence that enabled me to persevere. I am also grateful to my thesis readers, Dr. Vic Syed and Capt David King, for their support.

This thesis would not have been possible without the cooperation and computer time provided by the following people and organizations: ASD for the use of its CDC Cyber 750; Mr. Ron Berger of AFWAL/POTX for information on the IBM 370/155; Mr. Joe Hamlin and AFIT for the use of its DEC VAX 11/780; Mr. Dave McGrew and ASD for the use of its DEC PDP 11/60; Capt Wayne Warren and AFWAL/FIGX for the use of its DEC PDP 11/50; and Maj Larry Kizer and the Signal Processing Lab at AFIT for the use of its Cromemco Z-2D.

I would especially like to thank Capt Gary P. Route for the use of his Cray-1 assembly language listings and hardware manual. In addition, his previous experience with this subject saved me many hours and helped me explore the subject more thoroughly.

Mark A. Mehalic

Portions of this thesis were typed by Diane Katterheinrich.

Contents

List of Figures

List of Tables

Abstract

This study examines the effects of computer architecture on FFT algorithm performance. The computer architectures evaluated are those of the Cray-1, CDC Cyber 750, IBM 370/155, DEC VAX 11/780, DEC PDP 11/60, DEC PDP 11/50, and Cromemco Z-2D. The algorithms executed are the radix-2, mixed-radix FFT (MFFT), Winograd Fourier Transform Algorithm (WFTA), and prime factor algorithm (PFA).

The execution time of each algorithm for different sequence lengths is determined for each computer. The initialized WFTA is fastest on the Cray-1, the radix-2 is fastest on the CDC Cyber 750, and the PFA is fastest on the others. Then the number of assembly language instructions executed are determined for the following categories: data transfers, floating point additions and subtractions, floating point multiplications and divisions, and integer operations. The correlation coefficients between the number of assembly language instructions in each category and the algorithm execution speeds are determined for each computer. The average values of the correlation coefficients range from 0.8614 for the floating multiplications and divisions to 0.9792 for the data transfers. The values of the correlation coefficients are then related to the computer architectures.

The computer architectures are then compared against each other to determine what features are desireable in an FFT processor. The most desireable features are assembled into a proposed minimum computer architecture for efficient

FFT performance. The minimum architecture includes separate functional units, a cache memory, and separate floating point and integer registers. In addition, a method for deciding how to improve a given architecture is presented. The method is based on the correlation coefficients for that architecture. Guidelines for predicting FFT algorithm performance are given based on the known computer architecture. Floating point processors execute the radix-2 fastest, data transfer processors execute the PFA fastest, and vector processors execute the initialized WFTA fastest.

I. Introduction

Background.

A Fourier transform is a mathematical method of determining the frequency content of a given time domain signal representation. Fourier transforms are commonly found in many signal processing applications. But Fourier transforms, which are continuous functions, cannot be calculated directly on a digital computer. To overcome this problem, and take advantage of the processing power of digital computers, discrete Fourier transforms (DFTs) were developed. The equation defining the DFT is

$$X(k) = \sum_{n=0}^{N-1} x(n)\exp(-j2\pi nk/N) \qquad k=0,1,\ldots,N-1 \tag{1-1}$$

where x(n) consists of N samples of a finite length sequence and X(k) consists of N frequency components of the sequence's Fourier transform. Direct evaluation of (1-1) requires N^2 complex operations, where a complex operation is defined as a complex addition and a complex multiplication. Computationally efficient algorithms have been devised to reduce the number of operations required to calculate the DFT. These DFT algorithms have come to be known as Fast Fourier Transform (FFT) algorithms. The first, developed by Cooley and Tukey in 1965, required only $N\log_2 N$ complex operations (Cooley and Tukey, 1965). Since then, many new FFT algorithms have been devised.

The purpose of an FFT is to calculate the Fourier Transform of an input sequence as quickly as possible. With

the introduction of various computer architectures, some algorit ms have been reported to perform better than others on particular computers. But no specific and detailed comparisons have been made between the major FFT algorithms and the computer architectures. The process of optimizing an algorithm for a specific computer architecture, or conversely, optimizing a computer architecture for a specific Fourier Transform algorithm, is important in the design of dedicated FFT processors. Since FFT computations are required in many signal processing applications, the optimized design of a dedicated FFT processor will greatly increase throughput and decrease costs of signal processing systems.

Problem.

In the past, all FFT algorithm performance comparisons have been made on the basis of executing the programs on one particular computer architecture. This method gives a ranking of the algorithms but does not investigate the effects of the computer architecture on the algorithm performance. This study has three purposes. The first is to determine the performance rankings, based on execution speed, of each of the four most widely used algorithms as they are executed on different computer architectures. This will allow FFT users to select the fastest algorithm given the computer architecture available. The second is to determine the relationship between the computer architecture and the algorithm performance. This will provide information to those trying to optimize the implementation

of the algorithm or the architecture for maximum performance. The final purpose is to determine the minimum architecture required for efficient execution of FFT algorithms. This will provide a guideline for those designing or acquiring computer systems for the purpose of performing FFTs.

Scope.

This study examines the execution of four algorithms on seven different computers. The algorithms are a radix-2 algorithm, the Singleton mixed-radix algorithm, the Winograd Fourier Transform algorithm, and the Burrus prime factor algorithm. The seven computers are a Cray-1, a CDC Cyber 750, an IBM 370/155, a DEC VAX 11/780, a DEC PDP 11/60, a DEC PDP 11/50, and a Cromemco Z-2D. The algorithms are evaluated for a selected set of sequence lengths: 512, 1024, and 2048 for the radix-2; 504, 630, 1008, 1260, and 2520 for all other algorithms. These sequence lengths provide representative samples, and the information found can be extrapolated to other sequence lengths. The sequence lengths were chosen so that those for the radix-2 are as close as possible to the other algorithms, thus allowing a direct comparison between all the algorithms. This study presents the execution time of each of the algorithms on each of the computers. In addition, a count of the various assembly language instructions was made for each of the algorithms. This study does not try to determine which algorithm is the best, or which computer architecture is the

best, but determines which algorithms match which architectures and also determines which hardware features are needed for fast execution of any FFT algorithm.

Assumptions.

This study assumes that the CPU clocks are as accurate as published in the manufacturer's manuals. Also, the impact of the operating system overhead is negligible because the implementation of the algorithm is a CPU intensive job and each implementation was executed either as the only job in the system or at a time of minimal activity. In addition, this study assumes that the manufacturer's published instruction timings are accurate.

Methodology.

The four major FFT algorithms are compared to each other on different computer architectures. The execution speed, memory requirements, and instruction counts for the four algorithms are compared. The algorithms studied are a decimation-in-time radix two algorithm, the Singleton mixed radix algorit m (MFFT), the Winograd Fourier Transform algorithm (WFTA), and the Burrus prime factor algorithm (PFA). Each algorithm is studied for different sequence lengths on seven different computers: a Cray-1, a CDC Cyber 750, an IBM 370/155, a DEC VAX 11/780, a DEC PDP 11/60, a DEC PDP 11/50, and a Cromemco Z-2D. The Cray-1 and CDC Cyber 750 were chosen as examples of scientific mainframe computers. The IBM 370/155 was chosen as an example of a general purpose mainframe computer. The DEC VAX 11/780 was chosen as an example of a new generation super minicomputer.

The DEC PDP 11/60 and PDP 11/50 were chosen as examples of minicomputers. The Cromemco Z-2D is an example of a microcomputer. Comparisons are made among the results and a determination made as to what hardware features are important for which algorithms.

The execution speed of each of the algorithms was determined by using the FORTRAN library subroutine calls to the real-time clock. The clock was called at the beginning of the algorithm and initialized to zero. At the end of the algorithm the clock was called again and the elapsed CPU time since the start of the algorithm determined. For the Cromemco Z-2D, which does not have a complete FORTRAN subroutine library, the execution time was determined by manually timing the algorithm with a stopwatch. For the DEC VAX 11/780, which also does not have a complete FORTRAN subroutine library, the execution time was determined by writing a subroutine in the C programming language which accessed the real time clock and could be called from the FOR RAN program. These times are compared and ranked for the different algorithms on each computer.

The instruction counts for each computer were determined with the help of the FORTRAN compilers. All of the FORTRAN compilers had an option for generating an assembly language listing. Once this listing was obtained, the flow of execution was determined and the number of times an instruction was executed was counted by hand. Once the number of different types of instructions was known, the

execution speed for other sequence lengths can be predicted, if the architecture and instruction cycle times are known.

Presentation.

Chapter 2 presents a review of the current literature. This literature review summarizes the current research efforts in the areas of FFT algorithm development, FFT processor architecture, and the relationship between the two. Chapter 3 analyzes each of the four algorithms in detail. The radix-2, MFFT, WFTA, and PFA are introduced and the theory of operation explained. The development of the number of times each section of the algorithm is executed is presented. Chapter 4 describes the computer architecture of each of the seven computers used. The hardware features are described, as well as the operating system and compiler used to execute the FFT algorithms. Chapter 5 presents the results of executing each of the four algorithms on each of the seven computers. The execution speeds for each of the algorithms and sequence lengths is presented, as well as the instruction counts. Correlation coefficients are determined between the execution speeds and the instruction counts. Chapter 6 compares the results of each computer against the other in order to determine the important architectural features. The differences in architecture are related to the differences in performance of the algorithms. Chapter 7 takes the important architectural features and combines them into the minimum architecture necessary for efficient FFT execution. This architecture could be used as a guideline in the design of dedicated FFT processors. Chapter 8

presents some guidelines for predicting which algorithm will be the fastest based on the computer architecture. Chapter 9 presents the conclusions drawn and recommendations for further study.

II. Literature Review

Fourier transforms have an important role in electrical engineering. With a Fourier transform, information in the time domain can be converted to the frequency domain. Often the frequency domain representation is more useful and easier to analyze. But calculating the Fourier Transform of an arbitrary time domain signal has always been a lengthy and difficult process. The appearance of digital computers, with their speed and ease of programming, has led to the development of many algorithms for computing a discrete Fourier transform (DFT). However, if signal processing requirements increase faster than hardware and algorithms are improved, many of these algorithms may not be usable due to their slow speed or large memory requirements. Therefore, research is being conducted on evaluating the currently available algorithms as well as developing new and better algorithms

Algorithms

The introduction of digital computers initiated the search for DFT algorithms. The first algorithm developed was based on a sequence length which was a power of two (Cooley and Tukey, 1965). This algorithm reduced the number of complex operations from N^2 to $N\log_2 N$. However, by taking advantage of the symmetry of $\exp(-j2\pi k/N)$, the number of complex multiplications can be reduced to $(N/2)\log_2 N$, while the number of complex additions stays the same. Since then, other algorithms have been developed in which the sequence

length is not restricted to being a power of two. The newer algorithms are more flexible, but may be slower.

Singleton developed an algorithm which uses mixed radixes (Singleton, 1969). Singleton used an algorithm by Gentleman and Sande to develop a mixed radix algorithm which uses the twiddle factors in order (Gentleman and Sande, 1966). In addition, the number of complex multiplications is reduced from $(p-1)^2$ to $(p-1)^2/4$ for odd p, where p is a factor of the sequence length N. A final permutation is used to arrange the results in order.

Kolba and Parks developed a prime factor algorithm which is better than Singleton's if the sequence length can be factored in a particular way (Kolba and Parks, 1977). This algorithm is based on the idea of circular convolution, and uses the Chinese Remainder Theorem to map the one dimensional sequence into multiple dimensions. This algorithm was found to be faster than Singleton's mixed radix algorithm.

Most recently, Winograd developed and algorithm using a series of nested multiplications (Silverman, 1977). Winograd developed short DFT algorithms based on more efficient convolution algorithms using a reduced number of multiplications. This algorithm reduces the number of multiplications while not increasing the number of additions.

Research has been done in comparing these algorithms in the areas of number of multiplications and additions and the array storage requirements (Blanken and Rustan, 1982).

Blanken and Rustan found that the MFFT is the most flexible algorithm and uses less memory than either WFTA or PFA. Expressions for the array storage requirements for each of the three algorithms were determined. WFTA and PFA require fewer real operations than MFFT. Blanken and Rustan state that selection of the most efficient algorithm in terms of speed depends on the machine speed of real additions versus real multiplications. However, they did not determine what the dependence was.

Architecture

Most of the studies which deal with execution times have only looked at two of the algorithms at a time (Morris, 1978). Morris examined the WFTA and a radix-4 algorithm on both a DEC PDP 11/55 and an IBM 370/168. He found that the radix-4 was better than WFTA both in terms of speed and memory requirements. Morris does relate the algorithm performance to the computer architecture; however, since the architectures are similar, the effects of the architecture cannot readily be determined. He also begins to explore the effects of the FORTRAN compiler on the algorithm performance, and states that algorithms cannot always be judged strictly in terms of the number of arithmetic operations.

Lately though, more people have become interested in comparing these algorithms to each other on a single computer. Burrus and Eschenbacher compared all four major algorithms in terms of speed and memory requirements on a

DEC PDP 11/45 minicomputer (Burrus and Eschenbacher, 1981). They found the PFA to be faster than the MFFT, radix-4, or Winograd algorithms. Also, since the PFA uses short DFT modules, unused modules can be removed when a fixed sequence length is to be calculated, or additional modules can be added to increase the number of sequence lengths that can be transformed. However, only one computer was used. Thus, the effects of the computer architecture on the algorithm performance could not be determined. They also determined the number of real operations for each algorithm.

The recent studies also began to relate the structure of the algorithm to the hardware architecture of various common computers. Route began by comparing four algorithms on four different medium- to large-sized computers (Route, 1981). He then tried to relate the speed and memory requirements of each algorithm to the architecture of each computer. Route was able to rank the algorithms against each other and also between the different computers, using speed and memory requirements as the basic criteria. He found that the choice of algorithm is important in maximizing computer efficiency. Increasing efficiency could decrease the cost and effort required in performing a particular signal processing task. Route found that the ranking of algorithms by execution speed could be different on different computers. He found that WFTA was fastest on a Cray-1, radix-2 was fastest on a CDC Cyber 750, and PFA was fastest on an IBM 370/155 and DEC PDP 11/60. He then related the algorithm performance to the computer

architecture by determining the instruction cycle times and instruction counts for each algorithm and computer. Route emphasized the effect of data transfers on the algorithm performance.

Research is also being conducted on developing faster algorithms. Most of these algorithms are being tailored to the computer architecture on which they will run. However, once an algorithm has been developed for a particular architecture, it cannot be transferred easily to another computer. Chu and Burrus have developed an algorithm which can be executed on a distributed microprocessor system (Chu and Burrus, 1982). The algorithm is based on the PFA, and has been modified to take advantage of distributed arithmetic. The calculation of the DFT is converted to a convolution. The algorithm was programmed on a Z-80 microprocessor and was 10-20 times faster than a radix-2 algorithm and 2-5 times faster than a standard PFA algorithm.

Also, traditional algorithms, such as the radix-2 algorithm, are being studied and modified to try to improve their performance (Preuss, 1982). This type of improvement depends more on mathematical advances than on technological advances. Recent advances in software are also being used to improve performance (Johnson and Burrus, 1982). These advances use the latest software design techniques and higher order language capabilities to optimize the program.

Computer architecture will continue to change.

Likewise, the demands placed on their signal processing capabilities will continue to increase. With the availability of microcomputers, the current trend is towards using a microcomputer dedicated to calculating a discrete Fourier Transform. In order to increase the speed and throughput, and to decrease the cost and memory requirement, the algorithm must match the architecture. Usually, the matching of algorithm to architecture is difficult to predict theoretically. Thus the major source of information is the experimental results obtained by executing the algorithms on various computers.

III. FFT Algorithms

This chapter presents an analysis of each of the four FFT algorithms used in this study.

Background

The basic equation for the evaluation of a discrete Fourier transform is

$$X(k)=\sum_{n=0}^{N-1} x(n)\exp(-j2\pi kn/N) \tag{3-1}$$

where n is the time domain index, x(n) is the time domain sample corresponding to n, k is the frequency domain index, X(k) is the frequency domain sample corresponding to k, and N is the number of time domain samples which is also the number of frequency domain samples (Oppenheim and Schafer, 1975). The equation is usually simplified by letting

$$W_N=\exp(-j2\pi/N) \tag{3-2}$$

thus giving

$$X(k)=\sum_{n=0}^{N-1} x(n)W_N^{kn} \tag{3-3}$$

For a particular value of k, x(n) must be multiplied by W_N^{kn} a total of N times. If x(n) is complex, i.e., consists of a real and imaginary part, then each multiplication of x(n) by W_N^{kn} requires four real multiplications. Thus, the evaluation of X(k) for a particular value of k requires 4N real multiplications. Since there are N different X(k) terms to be evaluated, the total number of real multiplications which must be performed is $4N^2$. Likewise,

the number of real additions to be performed is N(4N-2).

Most approaches to improving the efficiency of a DFT algorithm exploit either the symmetry or the periodicity, or both, of the exponential term. The symmetry is shown by

$$W_N^{k(N-n)}=(W_N^{kn})^* \tag{3-3}$$

where $(\ )^*$ means the complex conjugate of the quantity enclosed. The periodicity is shown by the identity

$$W_N^{kn}=W_N^{k(n+N)}=W_N^{(k+N)n} \tag{3-4}$$

Cooley and Tukey were the first to develop an algorithm which greatly reduced the number of operations required to calculate a DFT (Cooley and Tukey, 1965). The fundamental principle used in reducing the number of operations is decomposing the sequence length into a number of smaller discrete transforms. The two methods of decomposing the sequence length are the decimation-in-time, where the time sequence x(n) is decomposed, and the decimation-in-frequency, where the frequency sequence X(k) is decomposed. These algorithms with reduced operations counts have the number of real computations roughly proportional to $N\log_2 N$ and are generally known as fast Fourier transforms (FFTs).

<u>Radix-2</u>

The first algorithm evaluated is the in-place decimation-in-time radix-2 algorithm developed by Rabiner and Gold (Rabiner and Gold, 1975), which is similar to the one developed by Cooley and Tukey (Cooley and Tukey, 1965), with a permutation, or bit reversal, performed on the input

sequence. For an N length sequence, the algorithm requires $(N/2)\log_2 N$ complex multiplications and $N\log_2 N$ complex additions. The bit reversal interchanges are performed (N-2**[m/2])/2 times, where [ ] means the smallest integer greater than or equal to the quantity enclosed and m equals $\log_2 N$. The number of times each section of the radix-2 algorithm is executed is listed in Table 3.1 (Route, 1981). These equations will be used in determining the number of times various assembly language instructions are executed.

Singleton's Mixed Radix

The second algorithm evaluated is the Singleton mixed-radix algorithm (MFFT) (Singleton, 1979), which is a decimation-in-frequency algorithm since it decomposes the frequency index instead of the time index. The algorithm decomposes the sequence length into square factors, odd prime integers, and square free factors of prime integers. Then, the appropriate short transforms are performed. The algorithm requires $N(p_1+p_2+...+p_m-m)+N(m-1)$ complex multiplications and $N(p_1+p_2+...+p_m-m)$ complex additions, where p_i is the ith factor of the sequence length and m is the number of factors (Oppenheim and Schafer, 1975). The number of times each section is executed is listed in Table 3.2 (Route, 1981).

The Singleton mixed radix algorithm is based on the method proposed by Cooley and Tukey (Cooley and Tukey, 1965). The sequence length N is factored into m different prime factors as

TABLE 3.1

Number of Executions - Radix-2 Program

Program Section	Number of Times Executed
Bit-Reversal Counter Subtractions	$\sum_{n=1}^{m-1} 2^{n-1} (m-n)$
Bit-Reversal Interchanges	$(N - 2^{(m/2)}) / 2$
Butterfly Computations	$(N / 2) \log_2 N$

(Route, 1981)

TABLE 3.2

Number of Executions - Mixed Radix Program

Program Section	Number of Times Executed
Initialize Difference Equations Constants	m
Radix-2 Butterfly Without Twiddle Factor	$n_1 n_2 \ldots n_{i-1}(n_i - 1)$
Radix-2 Butterfly With Twiddle Factor	$(N/n_i)\ (n_i - 1) - n_1 n_2 \ldots n_{i-1}(n_i - 1)$
Radix-2 Twiddle Factors Calculated by Library Calls	$\frac{TWCAL_i}{32}$
where $TWCAL_i = (KSPAN_i - 1) - \frac{(KSPAN_i - 1)}{2} - 1$	
Radix-2 Twiddle Factors Calculated by Difference Equations	$TWCAL_i - \frac{TWCAL_i}{32}$
Radix-4 Butterfly With Twiddle Factors	$(\ (N/n_i)\ (n_i - 1) - n_1 n_2 \ldots n_{i-1}\ (n_i - 1)\ )\ /\ 3$
Radix-4 Butterfly Without Twiddle Factors	$(n_1 n_2 \ldots n_{i-1}(n_i - 1)\ )\ /\ 3$
Radix-4 Twiddle Factors Calculated by Library Calls	$\frac{(KSPAN_i - 1)}{32}$
Radix-4 Twiddle Factors Calculated by Difference Equations	$(KSPAN_i - 1) - \frac{(KSPAN_i - 1)}{32}$
Odd Transform Input Coefficients Additions	$(\ (p-1)\ /\ 2)\ \cdot\ (N/p)$
Odd Transform Cosine and Sine Multiplications	$(\ (p-1)\ /\ 2)^2\ \cdot\ (N/p)$
Odd Transform Resultant Coefficient Additions	$(\ (p-1)\ /\ 2)\ \cdot\ (N/p)$

TABLE 3.2 CONTINUED

Program Section	Number of Times Executed
Odd Factor Twiddle Factor Multiplications	$(N/n_i)\ (n_i-1)\ -n_1 n_2 \cdots n_{i-1}\ n_{i-1}(n_i-1)$
Odd Factor Unique Twiddle Factor Calculations	$(KSPAN_i-1)\ (n_i-1)$
Odd Factor Twiddle Factors Calculated by Library Calls	$\frac{(KSPAN_i-2)}{32}$
Odd Factor Twiddle Factors Calculated by Difference Equations	$(KSPAN_i-2) - \frac{(KSPAN_i-2)}{32}$
Square-Factor Pairwise Interchanges	$(N-n_1 n_2 n_3 n_4 n_5)\ /\ 2$
Digit-Reversal of Square-Free Indices	$(n_3 n_4 n_5-1)$
Temporary Store of Initial Square-Free Subsequence Coefficients	$(n_1 n_2)^2 \cdot PC$
Reordering of Square-Free Subsequence Coefficients	$(n_1 n_2)^2\ (n_3 n_4 n_5-PC-S)$

(Route, 1981)

m = Number of Factors

$N = n_1 n_2 n_3 \cdots n_{m-1} n_m$

$KSPAN_i = N/n_1 n_2 n_3 \cdots n_i$

$P = n_i$

n_i = Factor Currently Being Transformed

$$N = \prod_{i=1}^{m} n_i \tag{3-5}$$

The transform is then decomposed into m steps with N/n_i transformations of size n_i. For the radix-2 algorithm, n_i=2 for all values of i, but that need not be the case. The transform is computed in place, and a permutation is performed to rearrange the results in order.

Singleton proposed that the transform be expressed as a matrix multiplication

$$X = Tx \tag{3-6}$$

where x is the input sequence as a vector, X is the output sequence as a vector, and T is an n by n matrix of complex exponentials of the form

$$t_{kl} = \exp(-j2\pi kl/N) \tag{3-7}$$

The matrix T can then be factored into

$$T = PF_m F_{m-1} \cdots F_2 F_1 \tag{3-8}$$

where F_i is the transform step corresponding to the factor n_i of N and P is the permutation matrix. Then, each F_i matrix can be partitioned into N/n_i square submatrices of dimension N. This is the basis for Singleton's mixed radix FFT algorithm. In addition, we have $F_i = R_i T_i$, where R_i is a diagonal matrix of rotation, or twiddle, factors. When the rotation factors are separated into a matrix in this way, the trigonometric identities for the exponential form can be used to simplify the computations. For example, a multiplication by $\exp(-j2\pi)$ can be eliminated since $\exp(-j2\pi)=1$. The final matrix expression becomes

$$X=PR_m T_m R_{m-1} T_{m-1} \ldots R_2 T_2 R_1 T_1 x \qquad (3\text{-}9)$$

This equation is implemented in the FORTRAN code, a listing of which is contained in Appendix B.

Winograd Algorithm

The Winograd algorithm (WFTA) (McClellan and Nawab, 1979) is based on circular convolution, with the circular convolution of two sequences being equivalent to determining the N coefficients of a linear polynomial. The sequence is decomposed into a series of short transforms by the Chinese Remainder Theorem, with the multiplications nested inside the input and output additions. The number of times each section is executed is listed in Table 3.3 (Route, 1981).

In the Winograd algorithm, the number of multiplications is reduced while the number of additions remains at the typical FFT level of $N\log_2 N$ (Silverman, 1977). Like the Singleton mixed radix algorithm, the Winograd algorithm uses a matrix equation

$$X=D_N x \qquad (3\text{-}10)$$

where x and X are column vectors of the input and output values respectively and D_N is an N by N matrix with elements

$$D_N(i,r)=W_N^{ir}=W_N^{(ir)\mathrm{mod}N} \qquad (3\text{-}11)$$

The matrix D_N can be decomposed into the form

$$D_N=S_N C_N T_N \qquad (3\text{-}12)$$

where T_N is a J by N incidence matrix, C_N is a J by J

TABLE 3.3

Number of Executions - WFTA Program

Program Section	Number of Times Executed
Generate Multiplication Coefficients	ND1 • ND2 • ND3 • ND4
Input Index Mapping	N
Output Index Mapping	N
Permutation of Input Sequence	N
Permutation of Output Sequence	N
Perform "Nested" Multiplications	ND1 • ND2 • ND3 • ND4

(Route, 1981)

where NDi = number of multiplications for factor i.

diagonal matrix, and S_N is a N by J incidence matrix. The problem is to determine a solution to (3-12) where J is much smaller than N^2. Winograd used field theory to show that for J approximately equal to N, the number of multiplications is on the order of N. However, the number of additions is on the order of Nlog N, which is the same as a typical FFT. Winograd introduced a method of breaking a N length DFT into a nested series of smaller length DFTs. With this method, the number of multiplications does not grow as quickly with the sequence length N as in a typical FFT. However, this method does involve the reordering of the data both before and after processing. This reordering can be easily accomplished if N is factorable into mutually prime factors as described by the Chinese Remainder Theorem. The Chinese Remainder Theorem states that if an integer modulus is factored into two (or more) relatively prime factors $N=N_1 N_2$ and the residues of a smaller number n are evaluated modulus these two factors, i. e.,

$$n_1=\langle n\rangle_{N_1} \quad \text{and} \quad n_2=\langle n\rangle_{N_2} \tag{3-13}$$

then the original number can be reconstructed by the formula

$$n=\langle K_1 n_1+K_2 n_2\rangle_N \tag{3-14}$$

for suitably chosen K_i.

Using the short transforms, the matrix equation becomes

$$X'=(D_{N_1}*D_{N_{1-1}}*\ldots*D_{N_1})x' \tag{3-15}$$

Substituting equation (3-12) into (3-13) gives

$$X'=(S_{N_1} C_{N_1} T_{N_1} * S_{N_{1-1}} C_{N_{1-1}} T_{N_{1-1}} * \ldots * S_{N_1} C_{N_1} T_{N_1})x' \tag{3-16}$$

Using the property of Kronecker products (Thrall and Tornheim, 1957), we get the fundamental result of the WFTA.

$$X'=(S_{N_1} * S_{N_{1-1}} * \ldots * S_{N_1})(C_{N_1} * C_{N_{1-1}} * \ldots * C_{N_1})(T_{N_1} * T_{N_{1-1}} * \ldots * T_{N_1})x' \tag{3-17}$$

or equivalently

$$X'=S_N C_N T_N x' \tag{3-18}$$

The total number of multiplications is the product of the number of multiplications for the individual small N algorithms. In other words,

$$M_N=M_{N_1} M_{N_{1-1}} \ldots M_{N_1} \tag{3-19}$$

Since the diagonal elements of C are either strictly real or strictly imaginary, all multiplications can be performed using scalars. A listing of the program is included in Appendix C. The difference between the WFTA1 and WFTA2 algorithms used in this study is that the WFTA1 includes code to initialize some of the matrices for a particular sequence length.

Prime Factor Algorithm

The final algorithm evaluated is the prime factor algorithm (PFA) (Burrus and Eschenbacher, 1981), which is also based on circular convolution and uses the Chinese Remainder Theorem to decompose the sequence into short

transforms. However, the prime factor algorithm is more similar to the radix-2 and MFFT than to the WFTA. Unlike the MFFT and WF_A, the version of the PFA evaluated does not decompose the sequence length N, i. e., the factors of N must be specified. The algorithm can still be compared to the other three since the decomposition accounts for less than 1% of the execution time (Route, 1981). After the transform is complete, the output sequence is permuted, or unscrambled, into the proper order. The number of times each section is executed is listed in Table 3.4 (Route, 1981). The permutation is required because the PFA is an in-place algorithm. The reordering or unscrambling is done by the use of a generalization of the Chinese Remainder Theorem. The PFA is a completely new algorithm with no counterpart in the Cooley-Tukey approach.

Like all other FFTs, the PFA starts with the basic equation

$$c(k)=\sum_{n=0}^{N-1} x(n)W_N^{kn} \tag{3-20}$$

Then, the following change of variables is made.

$$n=\langle K_1n_1+K_2n_2\rangle_N \tag{3-21}$$

$$k=\langle K_3k_1+K_4k_2\rangle_N \tag{3-22}$$

and

$$\hat{x}(n_1,n_2)=x(\langle K_1n_1+K_2n_2\rangle_N) \tag{3-23}$$

$$\hat{c}(k_1,k_2)=c(\langle K_3k_1+K_4k_2\rangle_N) \tag{3-24}$$

TABLE 3.4

Number of Executions - PFA Program

Program Section	Number of Times Executed
Select Factors of N (N has m factors)	m
Initiate Transform Calculation for Factor	$\sum_{i=1}^{m} N/M_i$
Select Input Coefficients for Transform of Factor	$\sum_{i=1}^{m} (M_i-1)\ (N/M_i)$
Permutation of Output Sequence	N

(Route, 1981)

where $N = M_1 M_2 M_3 \ldots M_i \ldots M_m$

and M_i is current factor being transformed.

where $\langle\ \rangle_N$ means the quantity enclosed is evaluated modulo N. Thus, substituting these into equation (3-18) gives

$$\hat{c}(k_1,k_2)=\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}\hat{x}(n_1,n_2)W_N^{K_1K_3n_1k_1}W_N^{K_1K_4n_1k_2}W_N^{K_2K_3n_2k_1}W_N^{K_2K_4n_2k_2} \quad (3\text{-}25)$$

Now, if the K_is are chosen so that

$$\langle K_1K_3\rangle_N=N_2 \quad (3\text{-}26)$$

$$\langle K_2K_4\rangle_N=N_1 \quad (3\text{-}27)$$

$$\langle K_1K_4\rangle_N=\langle K_2K_3\rangle_N=0 \quad (3\text{-}28)$$

equation (3-23) reduces to

$$\hat{c}(k_1,k_2)=\sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1}\hat{x}(n_1,n_2)W_{N_1}^{n_1k_1}W_{N_2}^{n_2k_2} \quad (3\text{-}29)$$

which is the basic form of the PFA. A listing of the program implementing this algorithm is included in Appendix D. The unscrambling constant used to permute the sequence into the proper order is determined from

$$UNSC=\langle\sum_i(N/n_i)\rangle_N \quad (3\text{-}30)$$

where n are the factors of the sequence length N.

Now that each of the algorithms executed have been analyzed, the computer architectures on which they were executed will be explored.

IV. Computer Architectures

Before describing the architecture of each computer, the word architecture needs to be defined. A common definition of computer architecture is everything an assembly language programmer needs to know to write a program that will run. Thus, the available instructions and the number of registers would be part of the architecture, while data path widths and cache memory would not. However, for the purposes of this study, this definition is not sufficient. This study defines computer architecture as everything a FORTRAN programmer needs to know to write an FFT program that runs as fast as possible. With this definition, items such as cache memory and even the compiler are considered part of the computer architecture. This view of architecture encompasses the entire computer system. The main features of each computer architecture are listed in Table 4.1.

Cray-1.

The Cray-1 is a high speed scientific computer. It has 8 24-bit address registers and 8 64-bit scalar registers, as well as 8 64-element vector registers of 64 bits each. The CPU also contains a high speed buffer of 64 operand registers and 64 address registers, and four instruction buffers each containing 64 registers. The Cray-1 CPU has separate pipelined functional units for scalar addition, floating addition, and floating multiplication, each of which can operate in parallel with the others. The Cray-1

TABLE 4.1

Hardware Features of Each Computer Architecture

Computer	Registers	High Speed Buffer Memory	Functional Units
Cray-1	8 vector 8 scalar 8 address	254 instructions 64 operands 64 addresses	vector add floating multiply floating add scalar add
CDC Cyber 750	8 operand 8 address 8 index	12 instructions	floating multiply floating divide floating add integer add
IBM 370/155	4 floating 16 integer	8 kbytes	add decimal add and move
DEC VAX 11/780	16 integer	8 kbytes	integer add/multiply floating add/multiply
DEC PDP 11/60	6 floating 8 integer	2 kbytes	integer add/multiply floating add/multiply
DEC PDP 11/50	6 floating 8 integer	none	integer add/multiply floating add/multiply
Cromemco Z-2D	12 integer	none	integer add

real time clock used for algorithm timing has a resolution of 12.5 nanoseconds. A block diagram of the CPU is shown in Figure 4.1 (Cray Research, 2240004).

The FORTRAN source code was compiled using the CFT 1.09 compiler, which implements the FORTRAN 77 language. The programs were executed under the COS 1.09 operating system. The Cray-1 instruction set includes both vector and scalar operations. Table 4.2 contains the Cray-1 instruction timings.

CDC Cyber 750.

The CDC Cyber 750 is a high speed scientific computer. It has 8 operand registers, 8 address registers, and 8 index registers. Each operand register has a corresponding address and index register. Six of the register sets read from memory, while two write into memory. Placing an address into an address register causes the computer to initiate the desired fetch or store. The CPU has separate pipelined functional units for integer addition, floating addition, floating multiplication, and floating division that can operate in parallel. An instruction stack of 12 registers provides high speed buffer storage. According to CDC, the Cyber 750 real time clock has a resolution of 10 milliseconds. However, testing done by Route found the clock to be accurate to 1 millisecond. A block diagram of the system is in Figure 4.2 (Control Data Corporation, 1979).

The FORTRAN source code was compiled using the FTN 4.8

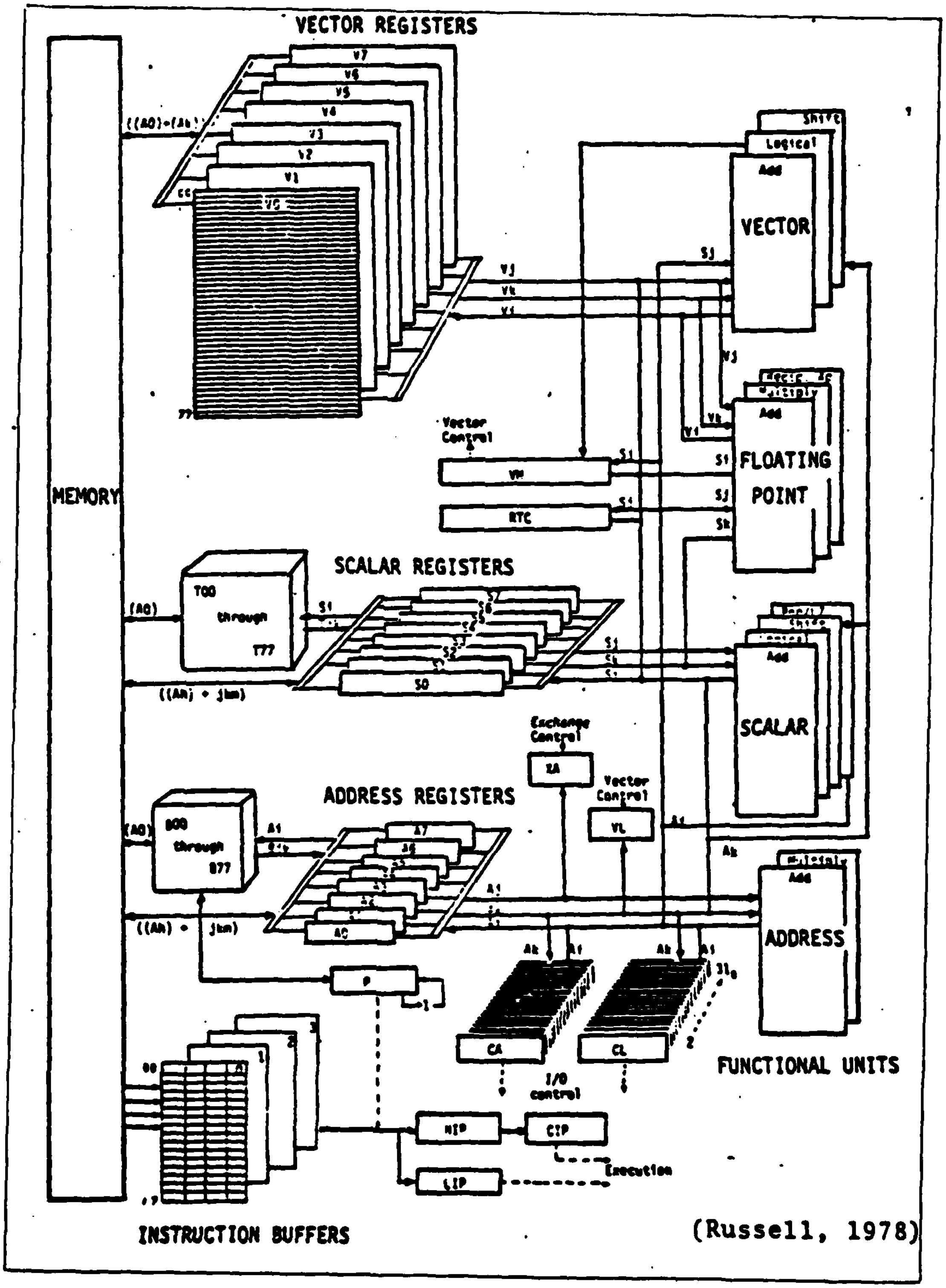


Fig 4.1. Cray-1 CPU Architecture

TABLE 4.2

Cray-1 Functional Unit Timings

Functional Unit	Execution Time in Nanoseconds
Address	
Add	25.0
Multiply	75.0
Scalar	
Add	37.5
Shift	25.0 or 37.5
Logical	12.5
Population/Leading Zero Count	50.0/37.5
Vector	
Add	37.5
Shift	50.0
Logical	25.0
Floating-Point	
Add	75.0
Multiply	87.5
Reciprocal Approximation	175.0

(Cray Research, 224004)

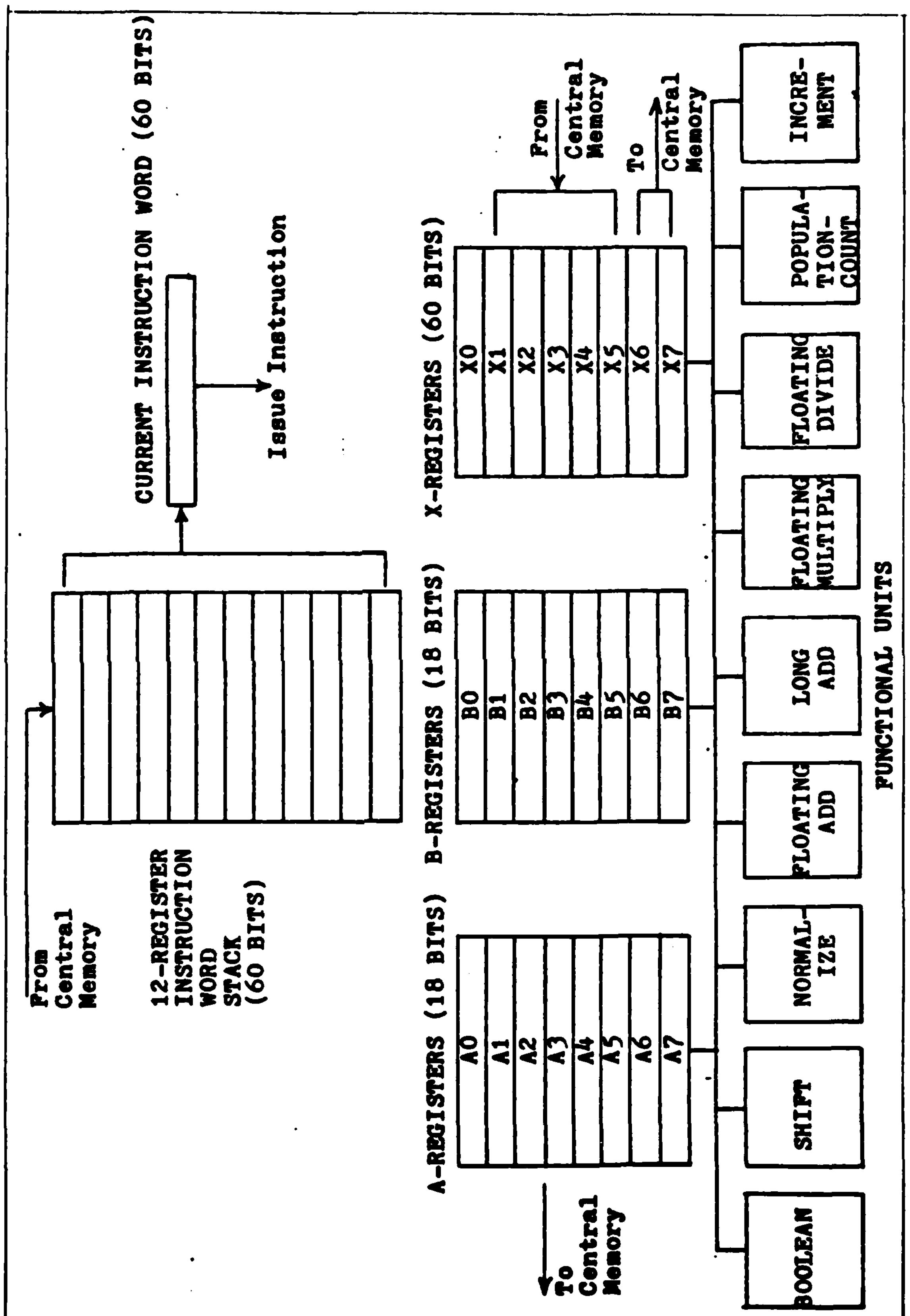


Fig 4.2. Cyber 750 CPU Architecture

compiler with option 2, which optimizes the execution time. The CDC compiler has three levels of optimization to improve the execution speed of the algorithm (Control Data Corporation, 1982). The compiler recognizes and replaces common expressions, removes invariant computations from within loops, retains frequently referenced variables in registers, and assigns frequently referenced variables to registers across loops. It also evaluates constant expressions at compile time, and simplifies subscript calculations by using additions instead of multiplications where possible. The compiler may reorder some of the operations to minimize the idle time of the functional units. The programs were run under the NOS/BE operating system. Table 4.3 lists the Cyber 750 instruction timings.

IBM 370/155.

The IBM 370/155 is a general purpose mainframe computer. It has 16 integer and address registers and 4 floating point accummulators. The CPU can pre-fetch up to three instructions, and the fetch and execution cycles can be overlapped. High speed buffer storage of 8 kbytes decreases the average data transfer time. The IBM 370/155 real time clock has a resolution of 3.33 milliseconds. A block diagram of the system is in Figure 4.3 (IBM, 1972).

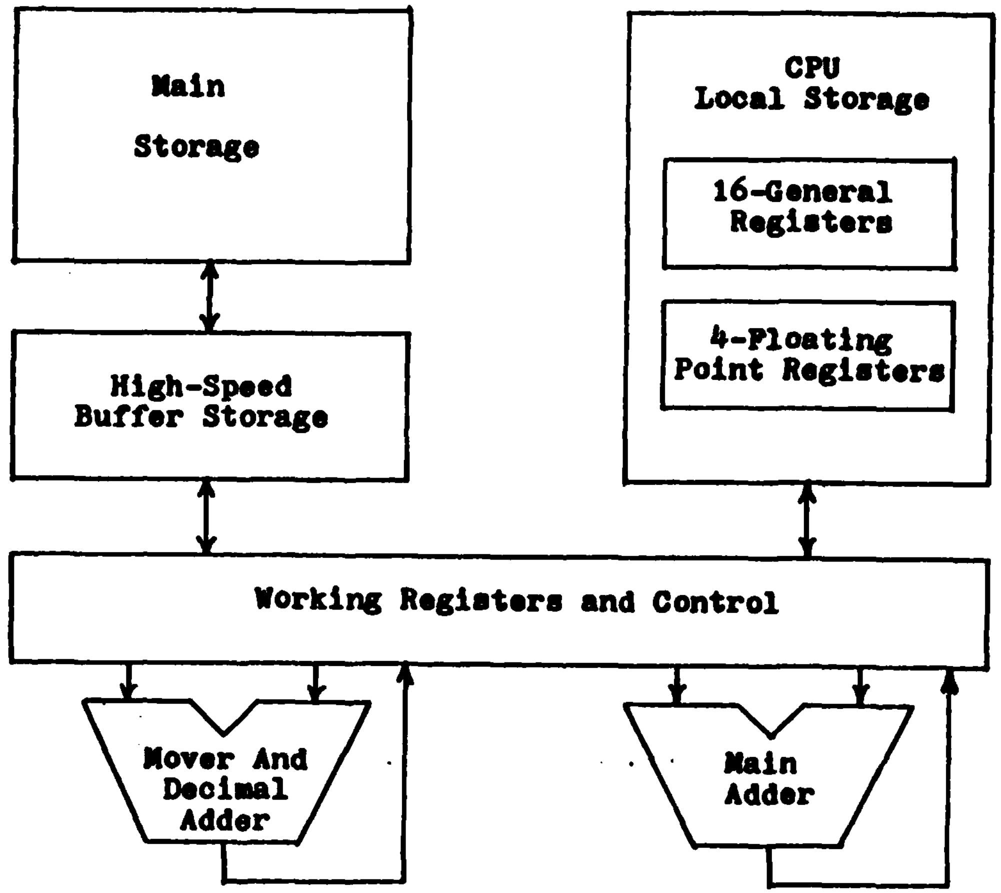
The source code was compiled using the FORTRAN H level 21.8 compiler with option 2, which gives the most optimized object code. The IBM compiler has three levels of optimization to improve the execution speed of the algorithms (IBM, 1974). The compiler recognizes and

TABLE 4.3

Cyber 750 Instruction Timings

Functional Unit	Execution Time in Nanoseconds
Boolean	50
Data Transfer Between X Registers	50
Shift	50
Normalize	75
Long Add	50
Floating Add	100
Floating Multiply	125
Floating Divide	500
Population Count	50
Increment	50
Operand Fetch (SA1 - SA5)	475
Operand Store (SA6 - SA7)	50
Branch Outside IWS	550
Branch Within IWS	75
Branch Condition Failed	50

(Control Data Corporation, 1979)



(IBM,1972)

Fig 4.3. IBM 370/155 Architecture

replaces common expressions, removes invariant computations from within loops, retains frequently referenced variables in registers, and assigns frequently referenced variables to registers across loops. It also simplifies subscript calculations by using additions instead of multiplications where possible. The programs were run under the OS/MVT version 21.8 F operating system. Table 4.4 lists the IBM instruction timings.

DEC VAX 11/780.

The DEC VAX 11/780 is a 32 bit super minicomputer. The computer was designed as an extension and improvement over DEC's PDP 11 series. It overcomes the main drawback of the PDP 11 series, namely, the program cannot be longer than 64 kbytes. In fact, the name VAX comes from the words virtual address extension. The VAX 11/780 is near the upper end of the performance scale of the VAX series of computers.

The VAX 11/780 has 16 32 bit registers (Digital Equipment Corporation, 1982). However, four of the registers are used by the operating system as a program counter, stack pointer, argument pointer, and frame pointer. The CPU contains three types of high speed buffer storage: a memory cache, an address translation buffer, and an instruction buffer. The memory cache contains 8 kbytes, is direct mapped, and is designed to achieve a cache hit ratio of 95%. When a cache miss occurs, 8 bytes are read from memory into the cache. The address translation buffer contains 128 of the most frequently used physical to virtual

TABLE 4.4

IBM 370/155 Instruction Timings

Instruction	Execution Time in Microseconds
Load Register	
LA	0.499
LCER	0.959
LER	0.844
LR	0.384
Load from Memory	
L	0.648
LE	0.993
Store to Memory	
ST	0.604 (2.104)[1]
STE	1.074 (2.344)[1]
Add Integer	
A	0.993
AR	0.499
Add Floating Point	
AE	2.749
AER	2.255
Multiply Floating Point	
ME	7.387
MER	6.778
Divide Floating Point	
DE	8.986
DER	8.952
Compare	
C	0.763
CR	0.384
Branch	
BC	0.844 (0.499)[2]
BCR	0.729 (0.384)[2]
BXLE	1.304 (0.959)[2]

[1]If the previous instruction was a store instruction, the execution time in parenthesis is used.

[2]If the branch conditions fail, the execution time in parenthesis is used.

(IBM, 1972)

address translations. The instruction prefetch buffer can hold 8 bytes and is always kept full by the CPU control logic. The VAX 11/780 used in this study contained the optional high performance floating point accelerator. The floating point accelerator is used on all floating point operations and can also be used on 32 bit integer multiply instructions. With the floating point accelerator, a floating add register to register can take as little as 800 nsec, while a floating multiply register to register can take as little as 1 usec. The CPU is microprogrammed, supports 32 interrupt priority levels, and has both a realtime clock and a time of year clock. The memory uses MOS technology and is expandable in 256 kbyte increments. An LSI 11 microcomputer is used as a console subsystem and for initial loading of the operating system. The VAX 11/780 real time clock has a resolution of 16.7 milliseconds. A detailed block diagram of the CPU is shown in Figure 4.4. Figure 4.5 contains the overall system block diagram.

The VAX 11/780 was designed for a multiprogramming environment with the inclusion of features such as rapid context switching, priority dispatching, virtual addressing, and memory management (Digital Equipment Corporation, 1981). The VAX 11/780 has a virtual address space of 4 Gbytes. However, half of this memory space is reserved for the operating system and overhead, leaving a user virtual address space of 2 Gbytes. The VAX supports five different integer data types: byte, word, longword, quadword, and octaword. In addition, there are four different floating

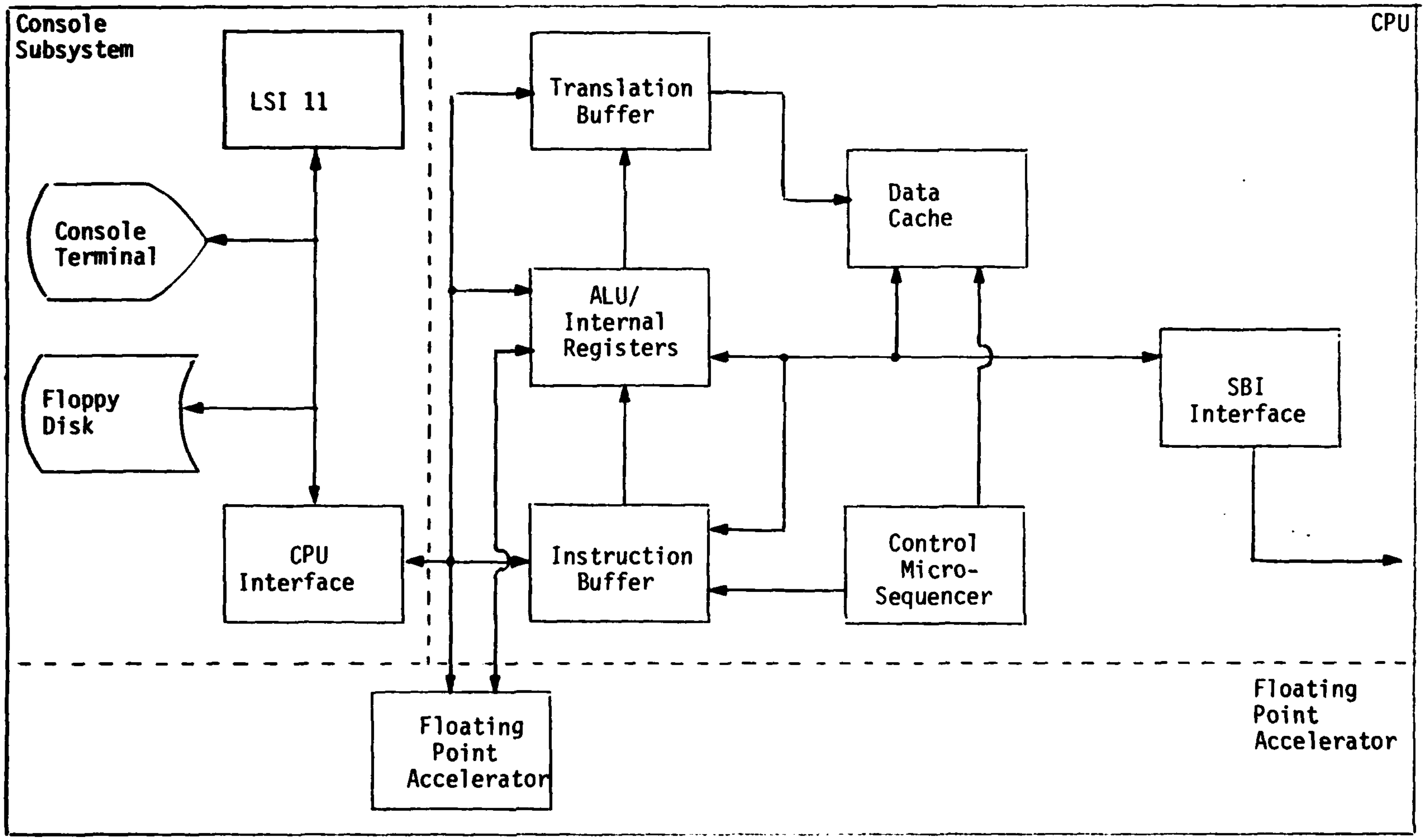


Figure 4.4. Expanded Block Diagram of DEC VAX 11/780 CPU

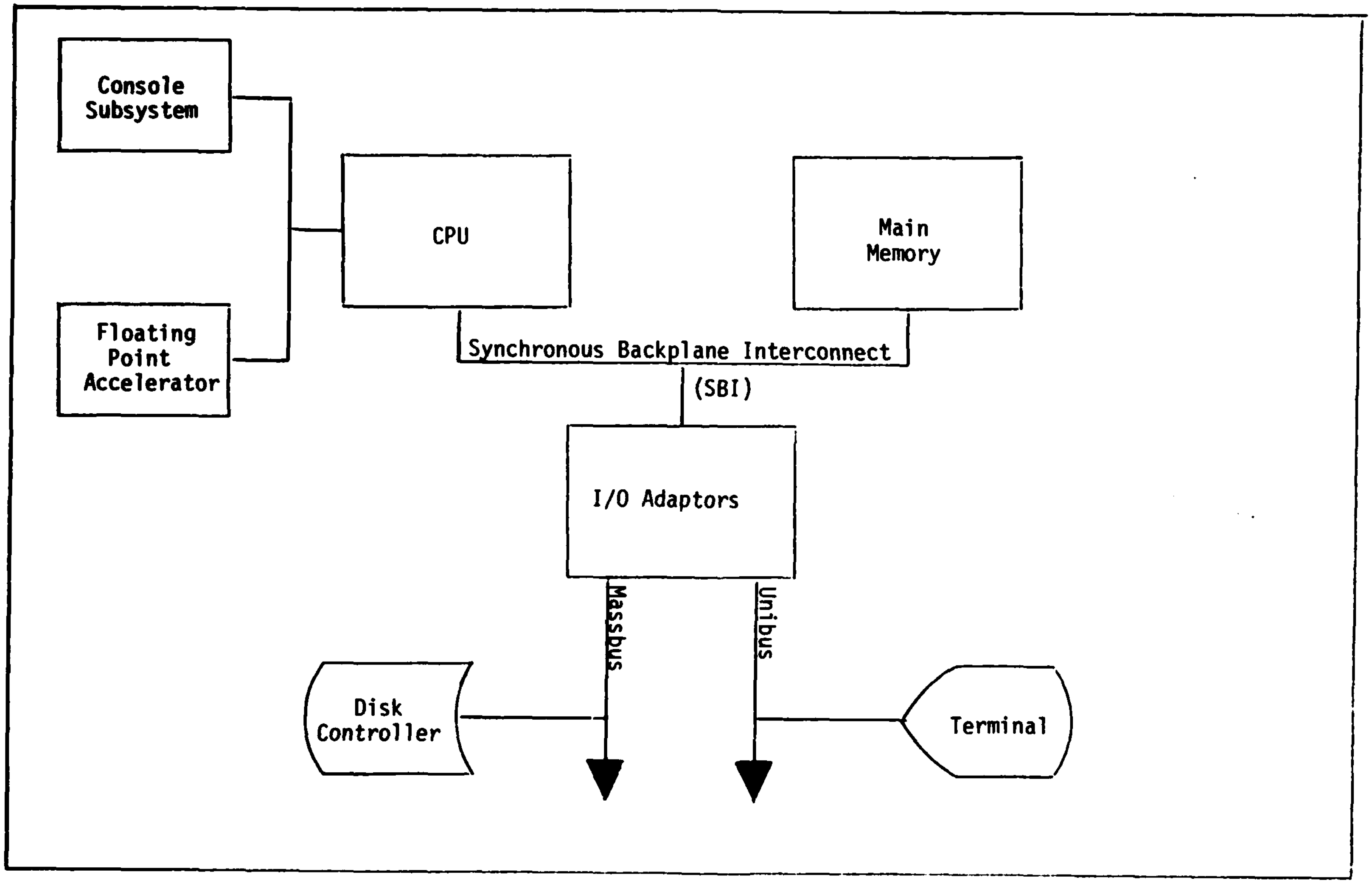


Figure 4.5. DEC VAX 11/780 System Block Diagram

point formats, each with varying degrees of precision. The VAX also has several other data types, such as packed decimal and character string. Like the PDP 11, the VAX has seven different operand addressing modes. The similarity of the VAX to the PDP 11 is deliberate. The VAX was designed so that PDP 11 programs could be executed on the VAX if desired.

The operating system under which the FFT programs were run is the Berkeley UNIX version 4.1 multi-user system (Kernighan, 1978). The operating system was written in the C programming language, and is designed to work best with the C language, but can be used with FORTRAN. The FFT programs were compiled using the FORTRAN 77 compiler f77 and the optional C language optimizer. The FORTRAN source code is first converted to C, optimized, and then compiled into machine code. The compiler also has an option for generating an assembly language listing. The instruction timings for the VAX are considered company confidential information and would not be released by DEC (Dixon, 1983). An attempt was made to determine the instruction timings experimentally by solving the system of equations determined from the results given in Chapter 5. However, since the VAX uses instruction overlap and a cache memory, the system is not linear, and no valid results could be obtained.

<u>DEC PDP 11/60.</u>

The DEC PDP 11/60 is a general purpose minicomputer. It has 8 integer registers, but only six are useable by the

program for operand storage, with the other two used as a program counter and stack pointer. The computer has the optional floating point processor for performing floating operations which can operate in parallel with the main CPU. The floating point processor contains 6 floating point accummulators, of which only 4 are memory accessable. The CPU contains 2 kbytes of high speed cache memory, and uses the DEC Unibus to transfer data in and out of memory. The PDP 11/60 real time clock has a resolution of 16.7 milliseconds. A block diagram is in Figure 4.6 (Digital Equipment Corporation, 1979).

The source code was compiled using the F4P version 3.0 compiler. The DEC compiler has a fixed level of optimization (Digital Equipment Corporation, 1981). The compiler recognizes and replaces common expressions, removes invariant computations from within loops, retains frequently referenced variables in registers, and assigns frequently referenced variables to registers across loops. It also evaluates constant expressions at compilation time. The programs were run under the DEC RSX-11M version 3.2 multiuser operating system. Table 4.5 lists the instruction timings.

<u>DEC PDP 11/50.</u>

The DEC PDP 11/50 is a general purpose minicomputer. For the purposes of this study, the PDP 11/50 is identical to the PDP 11/60 except that the PDP 11/50 has no cache memory and it has a slower version of the optional floating point processor. The PDP 11/50 real time clock has a

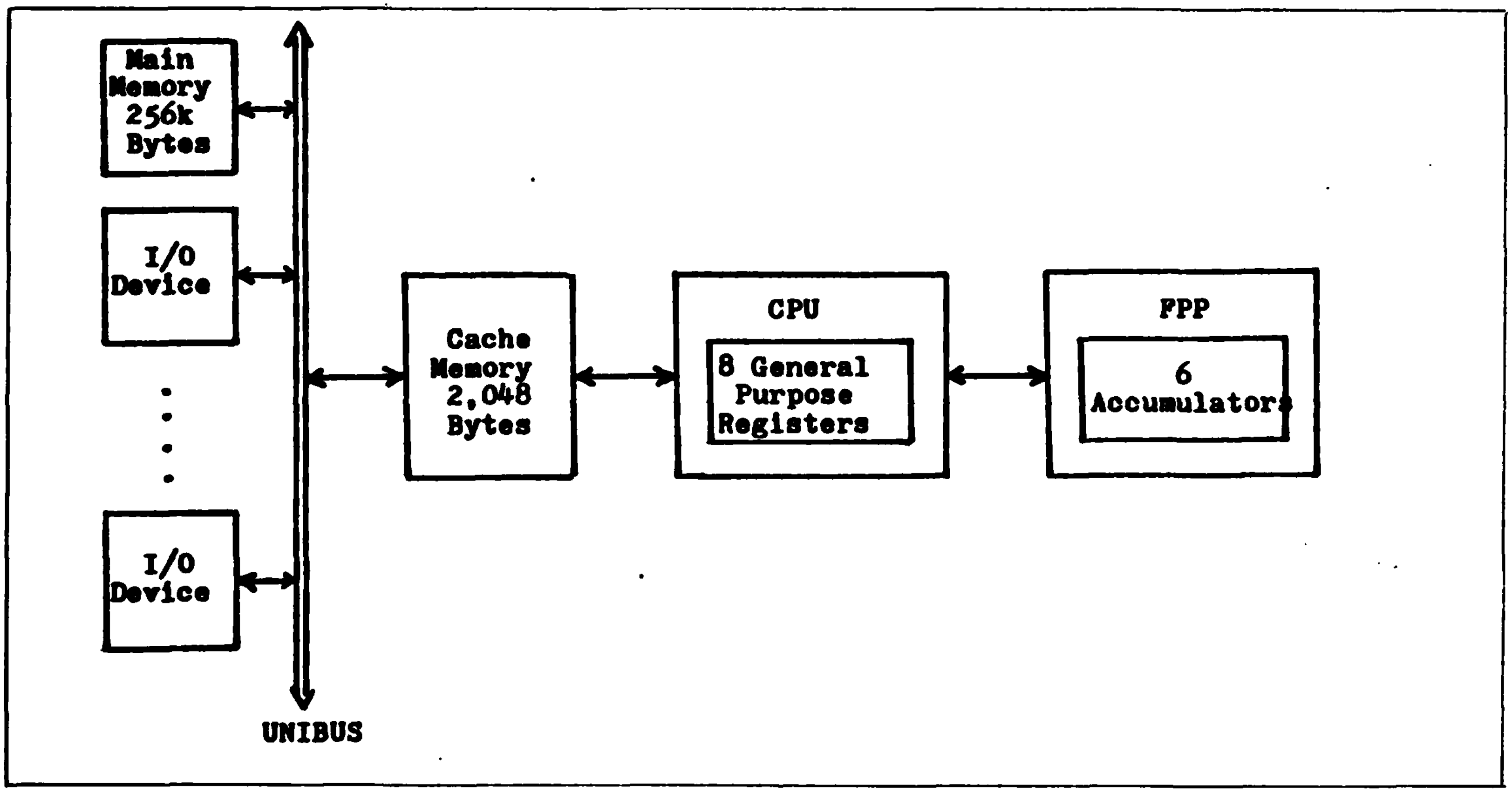


Fig 4.6. PDP 11/60 Architecture

TABLE 4.5

DEC PDP 11/60 Instruction Timings

Instruction	Execution Time in Microseconds
Floating Add/Subtract	2.55
Floating Multiply/Divide	2.72
Integer Operations	1.6
Integer Load/Store	1.11
Floating Load	0.79
Floating Store	2.34

resolution of 16.7 milliseconds. A block diagram of the architecture is shown in Figure 4.7 (Digital Equipment Corporation, 1976).

The source code was compiled using the F4P version 2.4 compiler. The compiler has a fixed level of optimization (Digital Equipment Corporation, 1981). It recognizes and replaces common expressions, removes invariant computations from within loops, retains frequently referenced variables in registers, and assigns frequently referenced variables to registers across loops. It also evaluates constant expressions at compile time. The programs were run under the RSX-11M version 3.1 multiuser operating system. Table 4.6 lists the instruction timings.

Cromemco Z-2D.

The Cromemco Z-2D is a general purpose microcomputer. It uses a Z-80 microprocessor with a 4 Megahertz clock as its CPU. The Z-80 contains 12 8-bit registers which can be combined into 6 16-bit registers. The CPU cannot perform multiplication or division and must rely on software routines for these functions. Memory is accessed through an S-100 bus. The Cromemco Z-2D has no real time clock. A block diagram of the architecture is shown in Figure 4.8 (Zilog, 1977).

The source code was compiled using the version 3.21 compiler supplied by Cromemco. The programs were run under the CDOS version 2.36 operating system, copyright 1980 by Cromemco, Inc. Table 4.7 lists the CPU instruction timings.

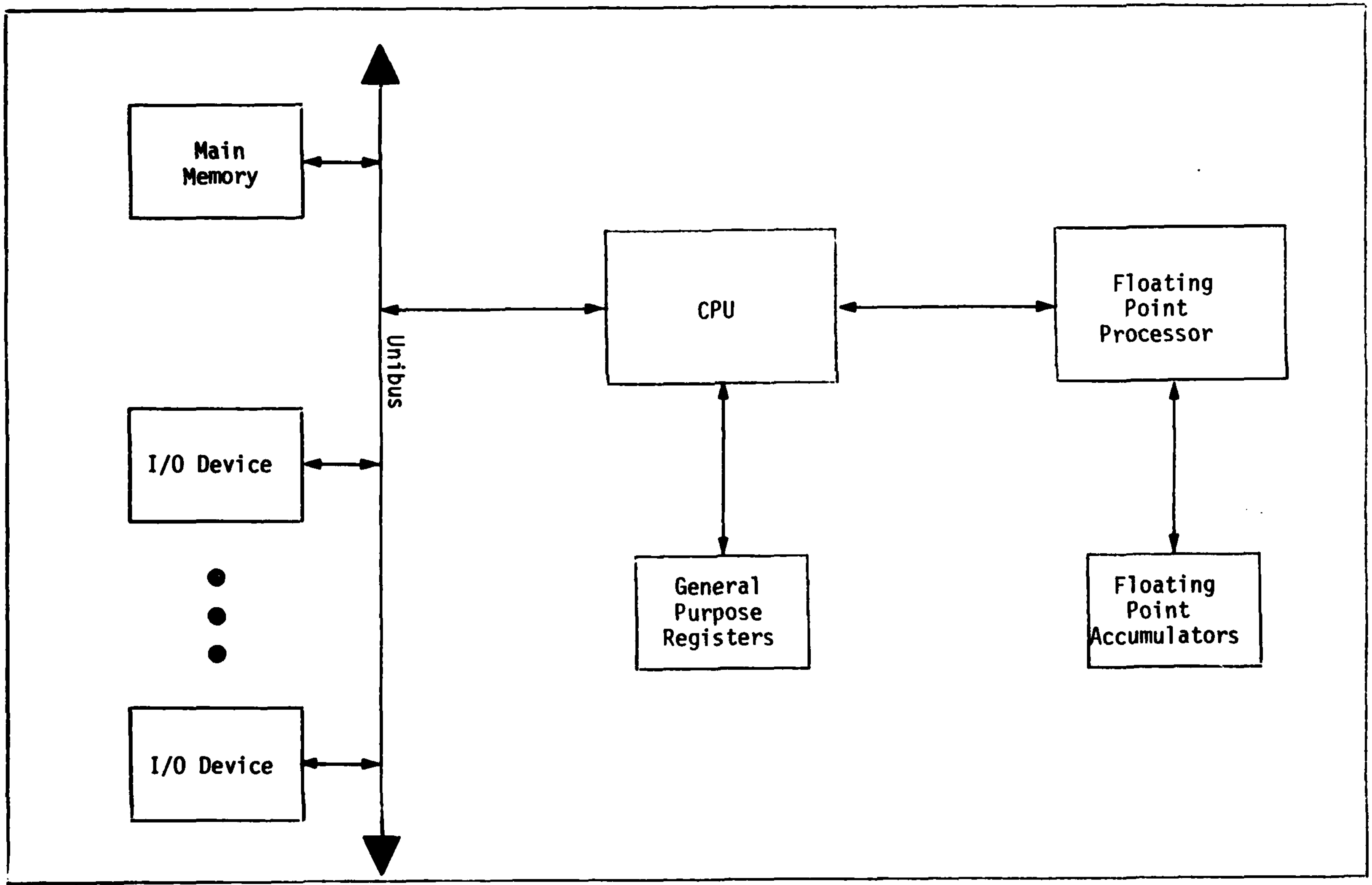


Figure 4.7. DEC PDP 11/50 Architecture

TABLE 4.6

DEC PDP 11/50 Instruction Timings

Instruction	Execution Time in Microseconds
Floating Load	4.20
Floating Store	3.58
Integer Load	0.95
Integer Store	0.95
Register Transfer	0.45
Memory Transfer	0.95
Branch	0.75
Compare	0.95
Increment	0.30
Decrement	0.30
Integer Add	0.60
Integer Subtract	0.60
Integer Multiply	3.60
Integer Divide	9.10
Floating Add/Subtract	5.66
Floating Multiply/Divide	7.61

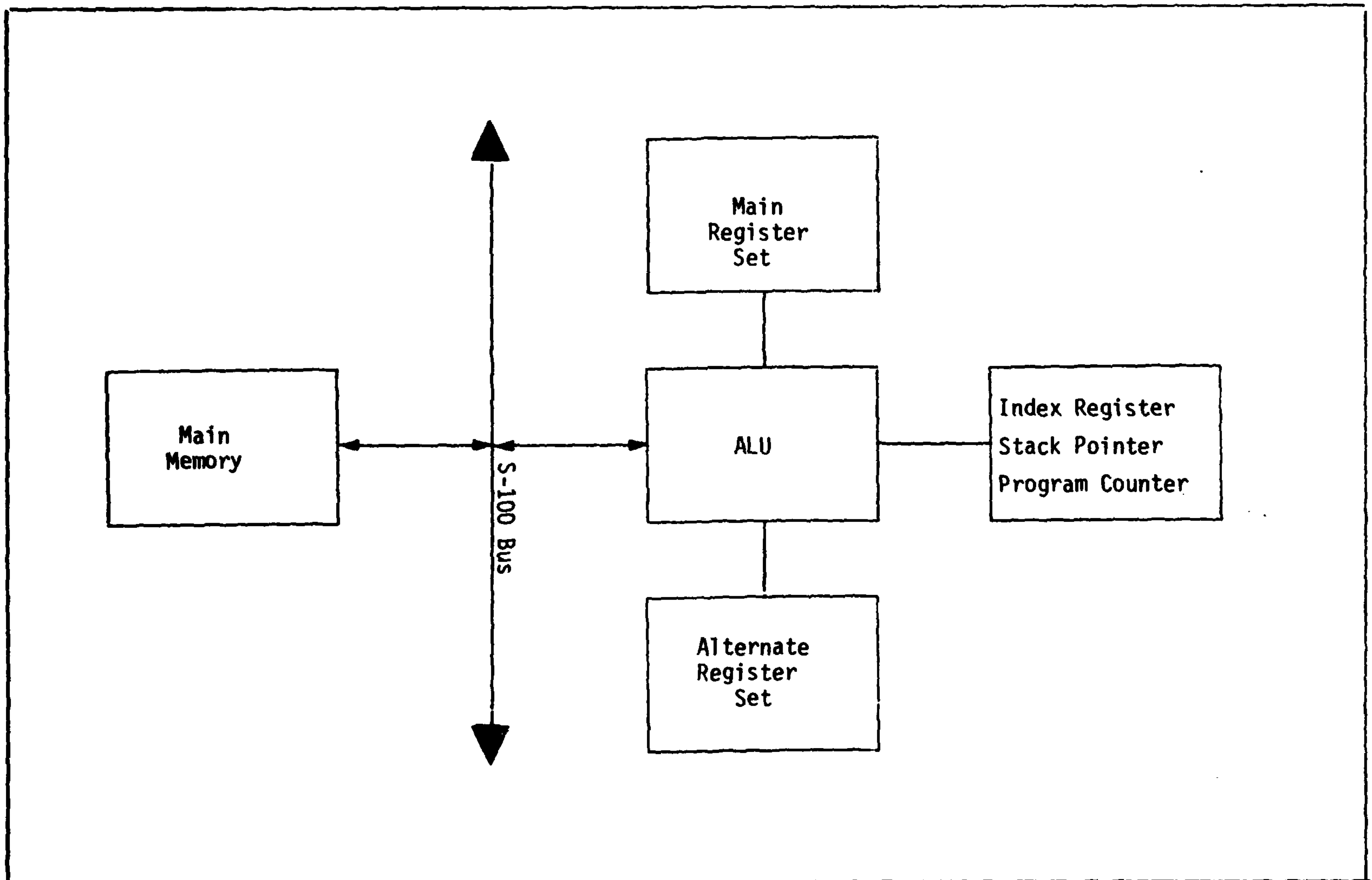


Figure 4.8. Cromemco Z-2D Architecture

TABLE 4.7

Cromemco Z-2D Instruction Timings

Instruction	Execution Time in Microseconds
Real Add	491.0
Real Multiply	1231.0
Data Transfer	4.0
Integer Add	1.0
Increment	1.0

The instruction timings for the floating point (real) operations were determined by executing a loop containing each instruction. Thus, the values for the floating point instructions were derived experimentally, while the others were provided by the manufacturer.

Now that an analysis of the algorithms has been presented, and each of the computer architectures has been described, the next chapter will present the results of executing the algorithms on the computer architectures.

V. Results

Each of the algorithms was run for different sequence lengths on seven computers: a Cray-1, a CDC Cyber 750, an IBM 370/155, a DEC VAX 11/780, a DEC PDP 11/60, a DEC PDP 11/50, and a Cromemco Z-2D. The radix-2 algorithm was executed for sequence lengths of 512, 1024, and 2048, while the other three were executed for sequence lengths of 504, 630, 1008, 1260, and 2520.

Methodology

The time required to perform a transform using each algorithm was determined for each sequence length. The times are average values obtained by repeated execution of the algorithm. The execution time measured was for the transform algorithm only and does not include the time to digitize the input data or to print results. The WFTA1 values include the time required for initialization, while the WFTA2 values do not. For all computers except the Cromemco Z-2D, the time was measured using the system realtime clock. The clock was initialized to zero at the beginning of the algorithm, and read at the end of the algorithm. The Cromemco Z-2D does not have a realtime clock, so the execution speed was measured with a stopwatch. Messages were printed on the screen at the beginning and end of the algorithms, and the time between these messages was measured. Repeated trials show this method to be accurate to within ±1 second, which is sufficiently accurate

considering the execution times of the algorithms on the Cromemco.

The memory requirements and data array sizes have been determined in other studies and the results are available in the literature (Burrus and Eschenbacher, 1981; Silverman, 1977; Kolba and Parks, 1977; Morris, 1978, Blanken and Rustan, 1982).

For all computers, an assembly language listing of the FORTRAN source code was obtained and analyzed. The number of instructions executed was determined for each of the different instruction types. These results were obtained by dividing the assembly language into the sections listed in Tables 3.1 through 3.4. Then the number of each type of instruction in that section was counted. The total number of that particualr instruction which was executed was obtained by multiplying the number of instructions for each section by the number of times each section was executed and then adding the results of each section. Typically, over 99.5% of the floating multiply/divide instructions were multiplies, while over 90% of the integer operations were additions or subtractions. The number of floating operations are dependent on the FFT algorithm, and are approximately equal to the number predicted by the equations given in the earlier section of this study, while the number of integer operations and data transfers are dependent on the compiler and the computer architecture, and are

different for each computer. However, all are dependent on the sequence length.

The percentage of time taken by each instruction category was determined by multiplying the number of each type of instruction by the effective instruction time. These individual results were then summed to find the total time. This time is not representative of the actual measured time because instruction overlap was assumed to be zero and some types of instructions (i.e., increments and decrements) were neglected. However, these values can be used for rough comparisons between algorithms and architectures. The relationship between the instruction counts, the execution speeds, and the computer architecture will be analyzed in the next chapter.

For each of the architectures, the correlation coefficients between the execution times and the instruction counts are determined. The correlation coefficients are calculated from the equation

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \qquad (5\text{-}1)$$

where x_i is the execution speed for a particular sequence length, y_i is the number of instructions for that instruction category and sequence length, $\bar{x}$ is the average execution time, $\bar{y}$ is the average instruction count, and N is the number of samples. The correlation coefficient is used to illustrate the relationship between two variables. It provides a method for evaluating the extent to which

variations in one variable are associated with variations in another variable and permits an appraisal of the closeness of the cause and effect relationship. A correlation coefficient can vary from -1 to +1. The closer the correlation coefficient is to +1, the greater the cause and effect relationship is between the variables. But in order for a correlation coefficient to be meaningful, the sample size must be large enough to eliminate chance effects. For most of the correlation coefficients calculated for this study, the sample size is twenty-three, which consists of three from the three sequence lengths of the radix-2 algorithm, five from the MFFT, five from the WFTA1, five from the WFTA2, and five from the PFA. A correlation coefficient was calculated between the execution time and each of the following instruction categories: data transfers, floating additions and subtractions, floating multiplications and divisions, and integer operations. For example, when calculating the correlation coefficient between the execution times and the data transfers, the execution speeds are the x terms, while the number of data transfers are the y terms. The correlation coefficients could indicate which instruction category is most closely related to the execution time.

Cray-1

Table 5.1 lists the execution speeds of each of the algorithms and sequence lengths on the Cray-1. Using a clock resolution of 12.5 nanoseconds and the minimum execution time of 2.73 milliseconds, the maximum percentage error is 0.5%. The correlation coefficients between the execution speeds and the four major instruction categories are:

floating multiply/divide	0.8848
floating add/subtract	0.9336
integer operations	0.9700
data transfers	0.9480.

Tables 5.2 through 5.6 list the number of instructions in each category for each algorithm and sequence length. The correlation coefficients range from 0.8848 for the floating multiplications and divisions to 0.9700 for the integer operations, with the correlation coefficient for the data transfers being 0.9480. The high correlation between the execution speed and number of data transfers is related to the instruction timings. An operand load requires 137.5 nsec, a store or register transfer requires 12.5 nsec, a floating multiply requires 87.5 nsec, and a floating add requires 75.0 nsec. The ratio of floating multiply speed to data transfer speed is 2.2. Figure 5.1 shows the percentage of execution time taken by each of the instruction types. The value used for the typical data transfer time was 106.25 nsec, which considers 75% of the data transfers to be loads, with the other 25% being stores and register transfers.

TABLE 5.1

Algorithm Execution Speeds in Milliseconds for Cray-1

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		5.75	7.95	2.73	3.71
512	4.25				
630		7.50	12.23	4.72	5.84
1008		10.55	17.84	6.77	7.77
1024	8.98				
1260		15.31	21.70	8.60	11.41
2048	18.97				
2520		32.30	42.34	17.25	23.03

TABLE 5.2

Cray-1 Radix-2 Results

Instruction Type	Sequence Length		
	512	1024	2048
Floating Multiply/Divide	11807	25615	55335
Floating Add/Subtract	13573	29702	64519
Integer Operations	19158	40921	86940
Data Transfers	112335	242663	520991
Total	156873	338901	727785

TABLE 5.3

Cray-1 MFFT Results

Instruction Type	Sequence Length				
	504	630	1008	1260	2520
Floating Multiply/Divide	13191	18403	23205	34319	82323
Floating Add/Subtract	16626	21713	33671	45035	103049
Integer Operations	36844	45568	62183	92974	205177
Data Transfers	97291	129923	178997	263228	580951
Total	163952	215607	298056	435556	971500

TABLE 5.4

Cray-1 WFTA1 Results

Instruction Type	Sequence Length				
	504	630	1008	1260	2520
Floating Multiply/Divide	15072	20184	30174	38856	76254
Floating Add/Subtract	14428	21932	34290	46384	99068
Integer Operations	52346	82254	114376	153053	302312
Data Transfers	209986	306803	450304	597900	1179628
Total	241832	431173	629144	836193	1657262

TABLE 5.5

Cray-1 WFTA2 Results

Instruction Type	Sequence Length				
	504	630	1008	1260	2520
Floating Multiply/Divide	1674	2484	3654	4860	9618
Floating Add/Subtract	14428	21932	34290	46384	99068
Integer Operations	26055	42569	61385	80996	165415
Data Transfers	86345	130407	200282	267037	539640
Total	128502	197392	299611	399277	813741

TABLE 5.6

Cray-1 PFA Results

Instruction Type	Sequence Length				
	504	630	1008	1260	2520
Floating Multiply/Divide	2524	4108	5810	8208	17668
Floating Add/Subtract	13388	18196	29548	38900	84088
Integer Operations	15492	24726	36909	47461	92812
Data Transfers	60313	88989	140144	175890	370048
Total	91717	136019	212411	270459	564616

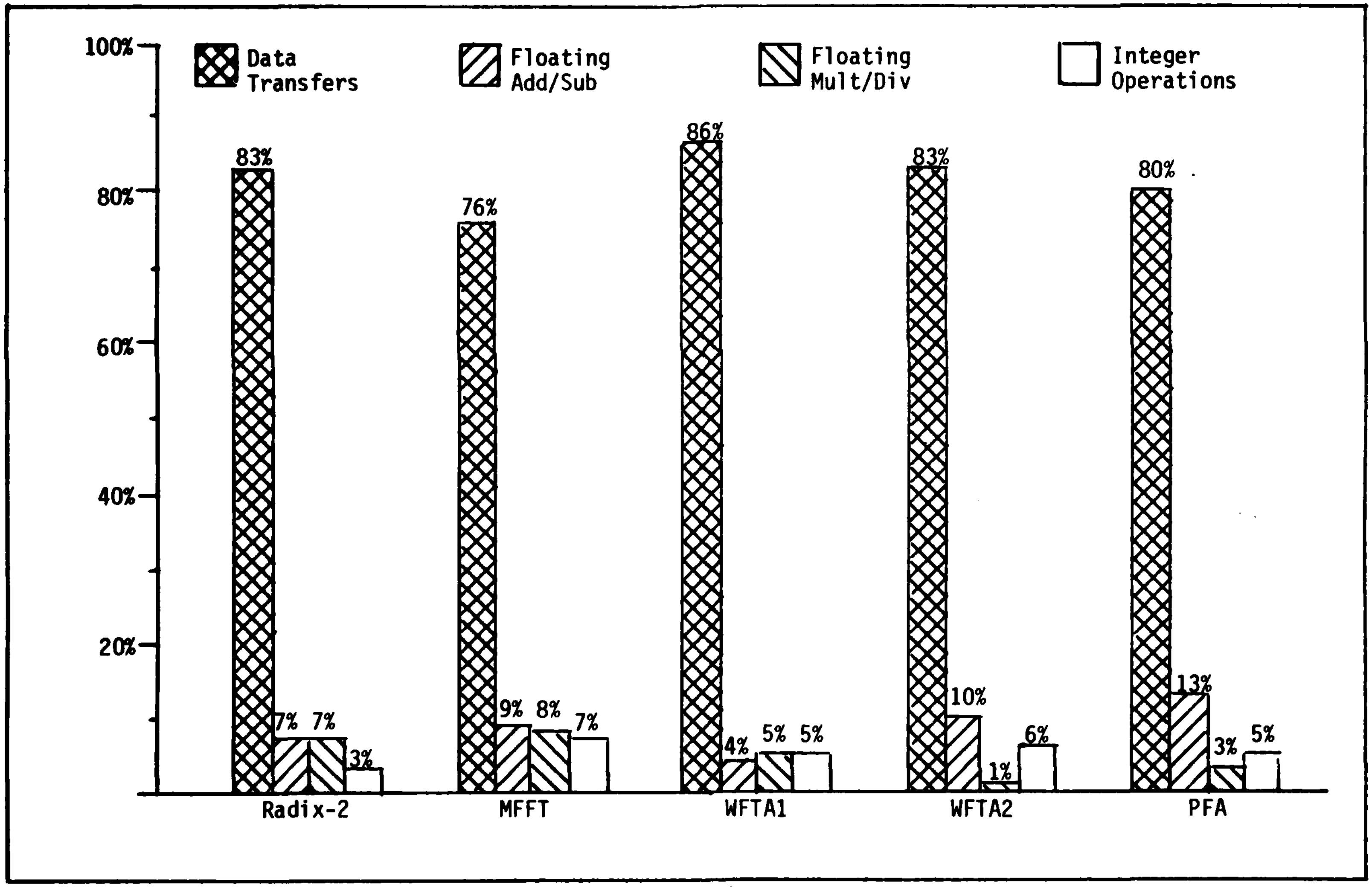


Figure 5.1. Percentage of Time per Instruction Category - Cray-1

Since over 99.5% of the integer operations were additions, the value of 25 nsec was used for the integer operations. Instructions other than those in the four major categories are neglected. The WFTA1 has the greatest percentage of time taken by data transfers with 86%, while the MFFT has the smallest with 76%. The WFTA2 has the fewest floating multiplies and is the fastest algorithm, while the WFTA1 has the most floating multiplies and is the slowest algorithm on the Cray-1. An examination of the assembly language shows that only the WFTA compiled using vector operations (Route, 1981), thus the WFTA2 had the shortest execution time. However, the compiler placed more floating multiplies and data transfers in the WFTA initialization routine than any other compiler, causing the WFTA1 to have the longest execution time. Thus, the availability of vector operations on the Cray-1 benefited the WFTA2.

CDC Cyber 750

Table 5.7 lists the execution speeds of each of the algorithms and sequence lengths on the CDC Cyber 750. Using a clock resolution of 1 millisecond and a minimum execution time of 10 milliseconds, the maximum percentage error is 10%. The object code produced by the optimizing compiler (option 2) on the Cyber 750 is three times faster than the code produced by the non-optimizing compiler (option 0). Throughout this study the optimized object code, when available, was compared for each of the computers. The correlation coefficients between the execution speeds and four major instruction categories are:

floating multiply/divide	0.7251
floating add/subtract	0.9187
integer operations	0.9569
data transfers	0.9988.

Tables 5.8 through 5.25 list the number of instructions in each category for each algorithm and sequence length. The values for the CDC Cyber 750 range from 0.7251 for the floating point multiplications and divisions to 0.9988 for the data transfers. The reason for the smaller correlation coefficient between the execution speed and the floating operations is the separate pipelined functional units of the Cyber 750, which increases the speed of the floating point operations but do not affect the data transfer rates. Thus the dependency of the execution speed on the floating operations is decreased. The radix-2 has the fewest data transfers and is the fastest algorithm on the Cyber 750,

TABLE 5.7

Algorithm Execution Speeds in Milliseconds for CDC Cyber 750

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		28	22	15	13
512	10				
630		36	30	21	20
1008		52	53	35	29
1024	24				
1260		76	55	41	43
2048	50				
2520		160	120	92	88

TABLE 5.8

CDC Radix-2 Results, N = 512

Instruction Type	Number of Times Executed		
	Bit-Reversal	Transform	Total
Float Add	---	14,846	14,846
Float Mult.	---	11,260	11,260
Float Div.	---	18	18
Load Operand	1,475	17,511	18,986
Store Operand	963	9,810	10,773
Register Transfer	1,975	55	2,030
Increment	1,541	10,553	12,094
Long Add	4,301	2,591	6,892
Branch	2,537	2,824	5,361
RJ ITOJ	1	9	10
RJ SIN	---	9	9
RJ COS	---	9	9
Others	2,027	14,891	16,918
Time (secs)	0.0016	0.0078	0.0094
Percent of Total Time	17	83	100

(Route, 1981)

TABLE 5.9

CDC Radix-2 Results, N = 1024

Instruction Type	Number of Times Executed Bit-Reversal	Transform	Total
Float Add	---	32,766	32,766
Float Mult.	---	24,572	24,572
Float Div.	---	20	20
Load Operand	3,011	38,003	41,014
Store Operand	1,987	21,595	23,583
Register Transfer	4,022	61	4,083
Increment	3,077	22,592	25,669
Long Add	8,650	5,155	13,805
Branch	5,095	6,153	11,248
RJ ITOJ	1	10	11
RJ SIN	---	10	10
RJ COS	---	10	10
Others	4,073	32,816	36,889
Time (secs)	0.0032	0.0166	0.0198
Percent of Total Time	16	84	100

(Route, 1981)

TABLE 5.10

CDC Radix-2 Results, N = 2048

Instruction Type	Number of Times Executed		
	Bit-Reversal	Transform	Total
Float Add	---	71,678	71,678
Float Mult.	---	53,244	53,244
Float Div.	---	22	22
Load Operand	6,019	82,047	88,066
Store Operand	3,971	47,204	51,175
Register Transfer	8,053	67	8,120
Increment	6,149	48,199	54,348
Long Add	17,335	10,279	27,614
Branch	10,213	13,322	23,535
RJ ITOJ	1	11	12
RJ SIN	---	11	11
RJ COS	---	11	11
Others	8,167	71,733	79,900
Time (secs)	0.0064	0.0357	0.0421
Percent of Total Time	15	85	100

(Route, 1981)

TABLE 5.11

CDC Singleton's Mixed-Radix Results, N = 504

Instruction Type	Number of Times Executed								
	Initialize	Transform Factor of				Odd Factor Rotation	Permu-tation	Total
		2	4	3	5	Odd			
Float Add	6	4,593	0	4,032	0	4,758	2,778	0	16,167
Float Mult.	47	2,894	0	1,344	0	2,605	4,902	77	11,869
Float Div.	17	1	0	0	0	4	5	2	29
Load Operand	118	9,354	6	6,388	0	13,071	8,416	5,764	43,117
Store Operand	84	4,376	0	2,857	0	8,528	4,226	3,163	23,234
Register Transfer	35	802	0	672	0	805	357	3,037	5,708
Increment	55	7	3	4	0	7,002	349	4,455	11,875
Long Add	73	3,644	3	2,018	0	6,074	4,442	4,651	20,905
Branch	37	966	3	507	0	1,166	1,251	1,862	5,792
RJ SIN	13	1	0	0	0	1	5	0	20
RJ COS	1	1	0	0	0	1	5	0	8
Others	99	5,390	0	4,368	0	6,576	4,075	991	21,449
Time (secs)	0.0002	0.0045	0	0.0027	0	0.0046	0.0036	0.0020	0.0176
Percent of Total Time	1	26	0	15	0	26	21	11	100

(Route, 1981)

TABLE 5.12

CDC Singleton's Mixed-Radix Results, N = 630

Instruction Type	Number of Times Executed								
	Initialize	Transform Factor of					Odd Factor Rotation	Permu-tation	Total
		2	4	3	5	Odd			
Float Add	5	2,085	0	5,040	4,034	5,946	4,427	0	21,537
Float Mult.	42	1,450	0	1,680	2,019	3,253	7,972	209	16,625
Float Div.	16	1	0	0	0	5	6	1	29
Load Operand	107	4,208	8	7,984	7,186	16,330	13,637	8,278	57,748
Store Operand	76	2,053	0	3,571	3,551	10,652	6,717	4,356	30,976
Register Transfer	34	421	0	840	2,016	1.005	480	4,428	9,224
Increment	49	3	4	4	0	8,750	468	6,189	15,467
Long Add	70	1,748	4	2,522	924	7,579	7,027	6,696	26,570
Branch	35	479	4	633	148	1,453	2,027	2,838	7,617
RJ SIN	11	1	0	0	0	1	6	0	19
RJ COS	1	1	0	0	0	1	6	0	9
Others	97	2,421	0	5,460	4,161	8,219	6,510	1,347	28,215
Time (secs)	0.0002	0.0020	0	0.0034	0.0028	0.0057	0.0059	0.0029	0.0229
Percent of Total Time	1	9	0	15	12	25	26	12	100

(Route, 1981)

TABLE 5.13

CDC Singleton's Mixed-Radix Results, N = 1008

Instruction Type	Number of Times Executed								
	Initialize	Transform Factor of					Odd Factor Rotation	Permu-tation	Total
		2	4	3	5	Odd			
Float Add	5	0	12,309	8,064	0	9,510	4,767	0	34,655
Float Mult.	38	0	5,759	2,688	0	5,197	9,164	2	22,848
Float Div.	14	0	7	0	0	4	2	2	29
Load Operand	108	0	23,242	12,772	0	26,109	15,843	6,778	84,852
Store Operand	76	0	13,377	5,464	0	17,030	7,195	3,871	47,013
Register Transfer	33	0	5,304	1,344	0	1,597	208	3,654	12,140
Increment	50	0	5	4	0	13,986	196	4,605	18,846
Long Add	62	0	3,333	3,536	0	12,140	7,354	6,347	32,772
Branch	32	0	1,794	762	0	2,324	2,314	2,156	9,382
RJ SIN	11	0	7	0	0	1	2	0	21
RJ COS	1	0	7	0	0	1	2	0	11
Others	86	0	12,156	8,736	0	13,128	7,102	1,617	42,825
Time (secs)	0.0002	0	0.0086	0.0054	0	0.0092	0.0070	0.0023	0.0327
Percent of Total Time	1	0	26	17	0	28	21	7	100

(Route, 1981)

TABLE 5.14

CDC Singleton's Mixed-Radix Results, N = 1260

Instruction Type	Initialize	Transform Factor of 2	4	3	5	Odd	Odd Factor Rotation	Permu-tation	Total
Float Add	6	7,540	0	10,080	8,066	11,886	9,303	0	46,881
Float Mult.	51	4,598	0	3,360	4,035	6,493	16,836	197	35,570
Float Div.	19	3	0	0	0	5	14	2	43
Load Operand	118	15,376	8	15,964	14,368	32,623	28,785	14,400	121,642
Store Operand	84	7,273	0	7,141	7,100	21,275	14,095	7,954	64,922
Register Transfer	35	1,269	0	1,680	4,032	1,995	956	7,682	17,649
Increment	57	7	4	4	0	17,480	944	11,229	29,725
Long Add	81	6,036	4	5,042	1,848	15,148	14,717	11,603	54,479
Branch	41	1,595	4	1,263	295	2,896	4,263	4,683	15,040
RJ SIN	13	3	0	0	0	1	14	0	31
RJ COS	1	3	0	0	0	1	14	0	19
Others	111	8,853	0	10,920	8,319	16,409	13,694	2,431	60,737
Time (secs)	0.0002	0.0074	0	0.0068	0.0056	0.0114	0.0127	0.0049	0.0490
Percent of Total Time	0	15	0	14	12	23	26	10	100

(Route, 1981)

TABLE 5.15

CDC Singleton's Mixed-Radix Results, N = 2520

Instruction Type	Number of Times Executed								
	Initialize	Transform Factor of 2	4	3	5	Odd	Odd Factor Rotation	Permutation	Total
Float Add	7	23,041	0	20,160	16,130	23,766	18,059	0	101,163
Float Mult.	60	14,638	0	6,720	8,067	12,973	32,746	414	75,618
Float Div.	22	9	0	0	0	2	27	2	65
Load Operand	127	46,698	8	31,924	28,732	65,203	56,331	29,544	258,567
Store Operand	90	21,780	0	14,281	14,156	42,515	27,333	16,288	136,443
Register Transfer	36	4,002	0	3,360	8,064	3,975	1,809	15,817	37,063
Increment	62	15	4	4	0	34,940	1,797	23,283	60,105
Long Add	91	18,048	4	10,082	3,612	30,268	28,414	23,720	114,239
Branch	46	4,762	4	2,523	547	5,776	8,273	9,749	31,680
RJ SIN	15	9	0	0	0	1	27	0	52
RJ COS	1	9	0	0	0	1	27	0	38
Others	125	26,950	0	21,840	16,635	32,789	26,577	5,037	129,953
Time (secs)	0.0002	0.0226	0	0.0136	0.0119	0.0228	0.0244	0.0101	0.1056
Percent of Total Time	0	21	0	13	11	22	23	10	100

(Route, 1981)

TABLE 5.16a

CDC WFTA Results, N = 504

Instruction Type	Number of Times Executed								
		Input Additions for Factor of							
	Initialize	2	3	5	7	8	9	16	Nested Mult.
Float Add	0	0	0	0	2,992	1,764	2,128	0	0
Float Mult.	2,214	0	0	0	1	1	3	0	1,584
Float Div.	20	0	0	0	0	0	0	0	0
Load Operand	8,657	1	1	1	2,738	1,422	2,745	1	2,382
Store Operand	4,739	0	0	0	2,385	1,399	2,011	0	1,584
Register Transfer	122	0	0	0	706	632	897	0	0
Increment	5,381	2	2	1	890	588	734	1	796
Long Add	16,239	2	2	1	100	34	145	1	2
Branch	10,247	2	2	1	90	72	65	1	793
Others	4,867	0	0	0	2,993	1,766	2,128	0	0
Time (secs)	0.0050	0	0	0	0.0013	0.0007	0.0012	0	0.0009
Percent of Total Time	36	0	0	0	9	5	9	0	7

(Route, 1981)

TABLE 5.16b

CDC WFTA Results, N = 504

Instruction Type	Number of Times Executed									
	Output Additions for Factor of									
	3	4	5	7	8	9	16	Perm1	Perm2	Total
Float Add	0	0	0	3,344	1,512	2,688	0	0	0	14,428
Float Mult.	0	0	0	1	1	3	0	3	3	3,814
Float Div.	0	0	0	0	0	0	0	0	0	20
Load Operand	1	1	1	3,530	1,359	3,305	1	1,861	1,866	29,873
Store Operand	0	0	0	2,297	1,336	2,123	0	1,092	1,023	19,989
Register Transfer	0	0	0	1,057	254	1,233	0	1,008	1,008	6,917
Increment	2	2	1	891	588	734	1	2,307	1,802	14,723
Long Add	2	2	1	100	34	145	1	146	84	17,041
Branch	2	2	1	90	72	65	1	576	576	12,658
Others	0	0	0	3,345	1,514	2,688	0	0	0	19,301
Time (secs)	0	0	0	0.0016	0.0006	0.0014	0	0.0006	0.0006	0.0139
Percent of Total Time	0	0	0	12	4	10	0	4	4	100

(Route, 1981)

TABLE 5.17a

CDC WFTA Results, N = 630

Instruction Type	Number of Times Executed								
		Input Additions for Factor of							Nested Mult.
	Initialize	2	3	5	7	8	9	16	
Float Add	0	1,260	0	3,168	3,740	0	2,660	0	0
Float Mult.	3,453	1	0	2	1	0	3	0	2,376
Float Div.	16	0	0	0	0	0	0	0	0
Load Operand	11,858	1,422	1	1,984	3,448	1	3,629	1	3,570
Store Operand	5,228	1,309	0	2,376	2,995	0	2,681	0	2,376
Register Transfer	617	0	0	792	882	0	1,121	0	0
Increment	9,928	798	2	1,593	1,142	1	1,134	1	1,192
Long Add	17,638	87	2	7	162	1	503	1	2
Branch	11,774	357	2	200	116	1	111	1	1,189
Others	4,797	1,260	0	3,168	3,741	0	2,660	0	0
Time (secs)	0.0064	0.0005	0	0.0010	0.0016	0	0.0015	0	0.0013
Percent of Total Time	34	3	0	5	8	0	8	0	7

(Route, 1981)

TABLE 5.17b

CDC WFTA Results, N = 630

Instruction Type	Number of Times Executed									
	Output Additions for Factor of									
	3	4	5	7	8	9	16	Perm1	Perm2	Total
Float Add	0	0	3,564	4,180	0	3,360	0	0	0	21,932
Float Mult.	0	0	2	1	0	3	0	3	3	5,848
Float Div.	0	0	0	0	0	0	0	0	0	16
Load Operand	1	1	2,380	4,438	1	4,329	1	3,595	3,628	44,288
Store Operand	0	0	1,584	2,885	0	2,821	0	1,660	1,311	27,226
Register Transfer	0	0	0	1,321	0	1,541	0	1,260	1,260	8,794
Increment	2	2	1,593	1,143	1	1,134	1	3,947	3,316	26,930
Long Add	2	2	7	162	1	503	1	714	400	20,195
Branch	2	2	199	116	1	111	1	986	986	16,155
Others	0	0	3,564	4,181	0	3,360	0	0	0	26,731
Time (secs)	0	0	0.0008	0.0020	0	0.0018	0	0.0011	0.0010	0.0190
Percent of Total Time	0	0	4	11	0	9	0	6	5	100

(Route, 1981)

TABLE 5.18a

CDC WFTA Results, N = 1008

Instruction Type	Initialize	Input Additions for Factor of							Nested Mult.
		2	3	5	7	8	9	16	
Float Add	0	0	0	0	6,732	0	4,788	4,788	0
Float Mult.	4,714	0	0	0	1	0	3	3	3,564
Float Div.	28	0	0	0	0	0	0	0	0
Load Operand	20,997	1	1	1	6,148	1	6,105	10,068	5,352
Store Operand	13,559	0	0	0	5,355	0	4,461	7,428	3,564
Register Transfer	131	0	0	0	1,586	0	2,017	1,386	0
Increment	10,404	2	2	1	1,990	1	1,574	3,638	1,786
Long Add	44,234	2	2	1	210	1	215	202	2
Branch	27,804	2	2	1	200	1	135	1,584	1,783
Others	13,709	0	0	0	6,733	0	4,788	4,788	0
Time (secs)	0.0129	0	0	0	0.0029	0	0.0026	0.0025	0.0019
Percent of Total Time	37	0	0	0	8	0	8	7	6

(Route, 1981)

TABLE 5.18b

CDC WFTA Results, N = 1008

Instruction Type	Number of Times Executed									
	Output Additions for Factor of									
	3	4	5	7	8	9	16	Perm1	Perm2	Total
Float Add	0	0	0	7,524	0	6,048	4,473	0	0	34,353
Float Mult.	0	0	0	1	0	3	3	3	3	8,298
Float Div.	0	0	0	0	0	0	0	0	0	28
Load Operand	1	1	1	7,930	1	7,365	8,178	3,373	3,378	78,902
Store Operand	0	0	0	5,157	0	4,713	5,188	2,100	2,031	53,556
Register Transfer	0	0	0	2,377	0	2,773	2,457	2,016	2,016	16,759
Increment	2	2	1	1,991	1	1,574	1,139	4,323	3,314	31,745
Long Add	2	2	1	210	1	215	167	146	84	45,697
Branch	2	2	1	200	1	135	73	1,080	1,080	34,086
Others	0	0	0	7,525	0	6,048	4,473	0	0	48,064
Time (secs)	0	0	0	0.0035	0	0.0031	0.0028	0.0012	0.0011	0.0345
Percent of Total Time	0	0	0	10	0	9	8	4	3	100

(Route, 1981)

TABLE 5.19a

CDC WFTA Results, N = 1260

Instruction Type	Number of Times Executed								
		Input Additions for Factor of							Nested
	Initialize	2	3	5	7	8	9	16	Mult.
Float Add	0	0	0	6,336	7,480	0	5,320	0	0
Float Mult.	6,455	0	0	2	1	0	3	0	4,752
Float Div.	15	0	0	0	0	0	0	0	0
Load Operand	20,294	1	1	3,964	6,858	1	6,989	1	7,134
Store Operand	9,246	0	0	4,752	5,965	0	5,131	0	4,752
Register Transfer	619	0	0	1,584	1,762	0	2,241	0	0
Increment	16,089	2	2	3,177	2,242	1	1,974	1	2,380
Long Add	31,808	2	2	7	272	1	573	1	2
Branch	20,522	2	2	398	226	1	181	1	2,377
Others	8,886	0	0	6,336	7,481	0	5,320	0	0
Time (secs)	0.0109	0	0	0.0020	0.0033	0	0.0030	0	0.0026
Percent of Total Time	30	0	0	6	9	0	8	0	7

(Route, 1981)

TABLE 5.19b

CDC WFTA Results, N = 1260

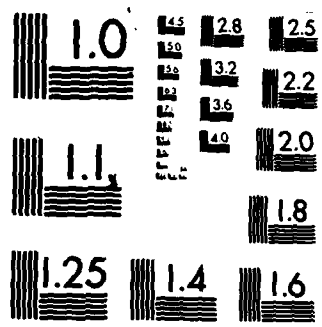
Instruction Type	Number of Times Executed									
	Output Additions for Factor of									
	3	4	5	7	8	9	16	Perm1	Perm2	Total
Float Add	0	5,040	7,128	8,360	0	6,720	0	0	0	46,384
Float Mult.	0	1	2	1	0	3	0	3	3	11,226
Float Div.	0	0	0	0	0	0	0	0	0	15
Load Operand	1	2,997	4,756	8,838	1	8,389	1	5,485	5,518	81,229
Store Operand	0	2,884	3,168	5,745	0	5,411	0	2,920	2,571	52,545
Register Transfer	0	317	0	2,641	0	3,081	0	2,520	2,520	17,285
Increment	2	1,462	3,177	2,243	1	1,974	1	6,467	5,206	46,401
Long Add	2	85	7	272	1	573	1	714	400	34,723
Branch	2	357	397	226	1	181	1	1,616	1,616	28,107
Others	0	5,042	7,128	8,361	0	6,720	0	0	0	55,274
Time (secs)	0	0.0017	0.0017	0.0039	0	0.0035	0	0.0018	0.0016	0.0360
Percent of Total Time	0	5	5	11	0	10	0	5	4	100

(Route, 1981)

AD-A138 465 EFFECTS OF COMPUTER ARCHITECTURE ON FFT (FAST FOURIER TRANSFORM) ALGORITH..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M A MEHALIC

2/3

UNCLASSIFIED DEC 83 AFIT/GE/EE/83D-47 F/G 12/1 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

TABLE 5.20a

CDC WFTA Results, N = 2520

Instruction Type	Number of Times Executed								
		Input Additions for Factor of							Nested
	Initialize	2	3	5	7	8	9	16	Mult.
Float Add	0	0	0	12,672	14,960	8.820	10,640	0	0
Float Mult.	12,473	0	0	2	1	1	3	0	9,504
Float Div.	17	0	0	0	0	0	0	0	0
Load Operand	40,314	1	1	7,924	13,678	7,090	13,709	1	14,262
Store Operand	20,455	0	0	9,504	11,905	6,979	10,031	0	9,504
Register Transfer	622	0	0	3,168	3,522	3,152	4,481	0	0
Increment	28,407	2	2	6,345	4,442	2,928	3,654	1	4,756
Long Add	69,613	2	2	7	492	154	713	1	2
Branch	44,326	2	2	794	446	356	321	1	4,753
Others	20,232	0	0	12,672	14,961	8,822	10,640	0	0
Time (secs)	0.0224	0	0	0.0040	0.0065	0.0037	0.0059	0	0.0051
Percent of Total Time	30	0	0	5	9	5	8	0	7

(Route, 1981)

TABLE 5.20b

CDC WFTA Results, N = 2520

Instruction Type	Number of Times Executed									
	Output Additions for Factor of									
	3	4	5	7	8	9	16	Perm1	Perm2	Total
Float Add	0	0	14,256	16,720	7,560	13,440	0	0	0	99,068
Float Mult.	0	0	2	1	1	3	0	3	3	21,997
Float Div.	0	0	0	0	0	0	0	0	0	17
Load Operand	1	1	9,508	17,638	6,775	16,509	1	9,265	9,298	165,976
Store Operand	0	0	6,336	11,465	6,664	10,591	0	5,440	5,091	113,965
Register Transfer	0	0	0	5,281	1,262	6,161	0	5,040	5,040	37,729
Increment	2	2	6,345	4,443	2,928	3,654	1	11,507	8,986	88,405
Long Add	2	2	7	492	154	713	1	714	400	73,471
Branch	2	2	793	446	356	321	1	2,876	2,876	58,674
Others	0	0	14,256	16,721	7,562	13,440	0	0	0	119,306
Time (secs)	0	0	0.0033	0.0078	0.0031	0.0069	0	0.0031	0.0029	0.0747
Percent of Total Time	0	0	5	10	4	9	0	4	4	100

(Route, 1981)

TABLE 5.21

CDC PFA Results, N = 504

Instruction Type	Number of Times Executed									
	PFA Control	Short DFT of Factor							Unscram-bling	Total
		2	4	5	7	8	9	16		
Float Add	0	0	0	0	5,184	3,276	4,704	0	0	13,164
Float Mult.	0	0	0	0	1,152	252	1,120	0	0	2,524
Float Div.	3	0	0	0	0	0	0	0	0	3
Load Operand	784	0	0	0	6,480	4,914	7,224	0	1,015	20,317
Store Operand	1,524	0	0	0	3,096	2,709	3,584	0	1.008	11,921
Register Transfer	0	0	0	0	1,728	1,134	1,736	0	0	4,598
Increment	2,294	0	0	0	0	0	0	0	3,726	6,020
Long Add	4,328	0	0	0	0	0	0	0	1,008	5,336
Branch	3,218	0	0	0	72	63	56	0	1,008	4,417
Others	1,346	0	0	0	5,184	3,276	4,704	0	0	14,510
Time (secs)	0.0011	0	0	0	0.0025	0.0018	0.0026	0	0.0005	0.0085
Percent of Total Time	13	0	0	0	29	21	31	0	6	100

(Route, 1981)

TABLE 5.22

CDC PFA Results, N = 630

Instruction Type	Number of Times Executed									
	PFA Control	Short DFT of Factor							Unscram-bling	Total
		2	4	5	7	8	9	16		
Float Add	0	1,260	0	4,284	6,480	0	5,880	0	0	17,904
Float Mult.	0	0	0	1,260	1,440	0	1,400	0	0	4,100
Float Div.	4	0	0	0	0	0	0	0	0	4
Load Operand	2,429	2,520	0	6,300	8,100	0	9,030	0	1,267	29,646
Store Operand	2,535	1,260	0	2,646	3,870	0	4,480	0	1,260	16,051
Register Transfer	0	0	0	1,638	2,160	0	2,170	0	0	5,968
Increment	4,947	0	0	0	0	0	0	0	5,018	9,965
Long Add	6,766	0	0	0	0	0	0	0	1,260	8,026
Branch	5,645	315	0	126	90	0	70	0	1,260	7,506
Others	1,952	1,260	0	4,284	6,480	0	5,880	0	0	19,856
Time (secs)	0.0026	0.0010	0	0.0025	0.0031	0	0.0033	0	0.0006	0.0131
Percent of Total Time	20	8	0	19	24	0	25	0	4	100

(Route, 1981)

TABLE 5.23

CDC PFA Results, N = 1008

Instruction Type	Number of Times Executed									
	PFA Control	Short DFT of Factor							Unscram-bling	Total
		2	4	5	7	8	9	16		
Float Add	0	0	0	0	10,368	0	9,408	9,324	0	29,100
Float Mult.	0	0	0	0	2,304	0	2,240	1,260	0	5,804
Float Div.	3	0	0	0	0	0	0	0	0	3
Load Operand	1,296	0	0	0	12,960	0	14,448	13,797	2,023	44,524
Store Operand	3,036	0	0	0	6,192	0	7,168	8,568	2,016	26,980
Register Transfer	0	0	0	0	3,456	0	3,472	3.150	0	10,078
Increment	4,318	0	0	0	0	0	0	0	7,382	11,700
Long Add	8,724	0	0	0	0	0	0	0	2,016	10,740
Branch	6,370	0	0	0	144	0	112	0	2,016	8,642
Others	2,730	0	0	0	10,368	0	9,408	9,324	0	31,830
Time (secs)	0.0020	0	0	0	0.0050	0	0.0053	0.0047	0.0010	0.0180
Percent of Total Time	11	0	0	0	28	0	29	26	6	100

(Route, 1981)

TABLE 5.24

CDC PFA Results, N = 1260

Instruction Type	Number of Times Executed									
	PFA	Short DFT of Factor							Unscram-	
	Control	2	4	5	7	8	9	16	bling	Total
Float Add	0	0	5,040	8,568	12,960	0	11,760	0	0	38,328
Float Mult.	0	0	0	2,520	2,880	0	2,800	0	0	8,200
Float Div.	4	0	0	0	0	0	0	0	0	4
Load Operand	3,573	0	6,615	12,600	16,200	0	18,060	0	2,527	59,575
Store Operand	5,055	0	2,520	5,292	7.740	0	8,960	0	2,520	32,087
Register Transfer	0	0	2,205	3,276	4,320	0	4,340	0	0	14,141
Increment	8,611	0	0	0	0	0	0	0	9,714	18,325
Long Add	14,073	0	0	0	0	0	0	0	2,520	16,593
Branch	10,971	0	315	252	180	0	140	0	2,520	14,378
Others	4,186	0	5,040	8,568	12,960	0	11,760	0	0	42,514
Time (secs)	0.0044	0	0.0030	0.0050	0.0063	0	0.0066	0	0.0012	0.0265
Percent of Total Time	17	0	11	19	24	0	25	0	4	100

(Route, 1981)

TABLE 5.25

CDC PFA Results, N = 2520

Instruction Type	Number of Times Executed									
	PFA Control	Short DFT of Factor							Unscram-bling	Total
		2	4	5	7	8	9	16		
Float Add	0	0	0	17,136	25,920	16,380	23,520	0	0	82,956
Float Mult.	0	0	0	5,040	5,760	1,260	5,600	0	0	17,660
Float Div.	4	0	0	0	0	0	0	0	0	4
Load Operand	5,861	0	0	25,200	32,400	24,570	36,120	0	5,047	129,198
Store Operand	10,095	0	0	10,584	15,480	13,545	17,920	0	5,040	72,664
Register Transfer	0	0	0	6,552	8,640	5,670	8,680	0	0	29,542
Increment	15,939	0	0	0	0	0	0	0	19,106	35,045
Long Add	28,568	0	0	0	0	0	0	0	5,040	33,608
Branch	21,623	0	0	504	360	315	280	0	5,040	28,122
Others	8,654	0	0	17,136	25,920	16,380	23,520	0	0	91,610
Time (secs)	0.0079	0	0	0.0100	0.0125	0.0088	0.0131	0	0.0025	0.0548
Percent of Total Time	14	0	0	18	23	16	24	0	5	100

(Route, 1981)

even though it does not have the fewest floating operations. For a sequence length of 1008, the WFTA1 has the most data transfers and is the slowest algorithm. For all other sequence lengths tested, the MFFT has the most data transfers and is the slowest algorithm. Thus the data transfer instructions are clearly more dominant than any of the floating operations. The correlation coefficient for the integer operations is large because integer operations are used to perform the address calculations for data transfers, and thus are closely related to the number of data transfers. The reason the data transfers are so important is clear from the instruction timings. Fetching an operand from memory requires 475 nsec while multiplying two floating point numbers requires only 125 nsec (Control Data Corporation, 1979). Figure 5.2 shows the percentage of execution time taken by each of the instruction types. The radix-2 has the smallest percentage of time taken by data transfers, which is consistent with the fact that it is the fastest, while the MFFT has the largest percentage of time taken by data transfers. Thus, the speeds of the algorithms on the Cyber 750 are limited mostly by the data transfer rate.

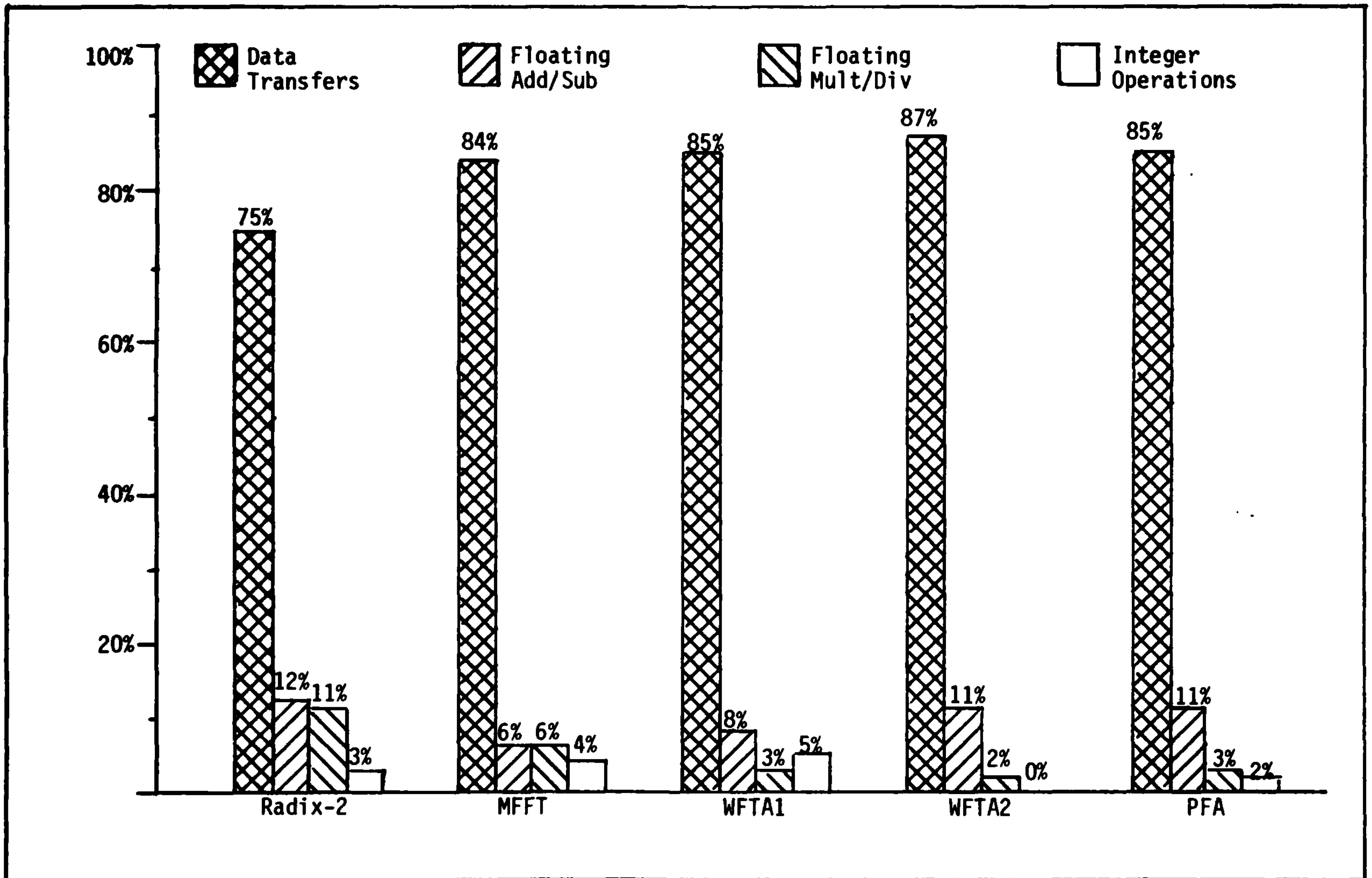


Figure 5.2. Percentage of Time per Instruction Category - CDC Cyber 750

IBM 370/155

Table 5.26 lists the execution speeds of each of the algorithms and sequence lengths on the IBM 370/155. Using a clock resolution of 3.33 milliseconds and a minimum execution time of 103 milliseconds, the maximum percentage error is 3.2%. The correlation coefficients between the execution speeds and the four major instruction categories are:

floating multiply/divide	0.8659
floating add/subtract	0.9522
integer operations	0.7760
data transfers	0.9507.

Tables 5.27 through 5.44 list the number of instructions in each category for each algorithm and sequence length. The correlation coefficients vary from 0.7760 for the integer operations to 0.9522 for the floating add/subtract. The execution speed is most closely related to the number of floating point additions and subtractions, which is exemplified by the fact that the PFA is the fastest algorithm on the IBM and has the fewest floating point additions and subtractions, while the MFFT is the slowest and has the most floating point additions and subtractions. Since the architecture has a multipurpose functional unit, the floating operations have a greater effect on the execution speed, whereas the high speed buffer memory decreases the dependence of the execution speed on the number of data transfers. Figure 5.3 shows that the WFTA1 has the largest percentage of execution time taken by data

TABLE 5.26

Algorithm Execution Speeds in Milliseconds for IBM 370/155

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		237	190	134	103
512	194				
630		307	280	203	147
1008		441	433	314	226
1024	404				
1260		657	533	408	314
2048	920				
2520		1423	1123	873	682

TABLE 5.27

IBM Radix-2 Results, N = 512

Instruction Type	Number of Times Executed			Percent of Total Time
	Bit-Reversal	Transform	Total	
Float Add	0	14,855	14,855	20
Float Multiply	0	11,260	11,260	44
Float Divide	0	9	9	0
Load Register	7	11,576	11,583	4
Load	2,730	13,928	16,658	8
Store	3,000	16,498	19,498	11
Integer Add	1,525	5,128	6,653	2
Integer Multiply	0	0	0	0
Integer Divide	503	9	512	2
Compare	1,524	520	2,044	1
Branch	3,048	2,842	5,890	3
Others	743	2,833	3,576	3
FIXPI#	1	9	10	1
SIN	0	9	9	1
COS	0	9	9	0
Time (secs)	0.0169	0.1724	0.1893	100
Percent of Total Time	9	91	100	

(Route, 1981)

TABLE 5.28

IBM Radix-2 Results, N = 1024

Instruction Type	Number of Times Executed			Percent of Total Time
	Bit-Reversal	Transform	Total	
Float Add	0	32,776	32,776	21
Float Multiply	0	24,572	24,572	45
Float Divide	0	10	10	0
Load Register	7	25,662	25,669	4
Load	5,545	29,812	35,357	8
Store	6,071	35,967	42,038	11
Integer Add	3,060	11,273	14,333	2
Integer Multiply	0	0	0	0
Integer Divide	1,014	10	1,024	2
Compare	3,059	1,033	4,092	1
Branch	6,118	6,173	12,291	3
Others	1,510	6,163	7,673	3
FIXPI#	1	10	11	0
SIN	0	10	10	0
COS	0	10	10	0
Time (secs)	0.0340	0.3741	0.4081	100
Percent of Total Time	8	92	100	

(Route, 1981)

TABLE 5.29

IBM Radix-2 Results, N = 2048

Instruction Type	Number of Times Executed			Percent of Total Time
	Bit-Reversal	Transform	Total	
Float Add	0	71,689	71,689	21
Float Multiply	0	53,244	53,244	45
Float Divide	0	11	11	0
Load Register	7	56,388	56,395	4
Load	11,096	63,616	74,712	8
Store	12,150	77,964	90,114	11
Integer Add	6,131	24,586	30,717	2
Integer Multiply	0	0	0	0
Integer Divide	2,037	11	2,048	2
Compare	6,130	2,058	8,188	1
Branch	12,260	13,344	25,604	3
Others	3,029	13,333	16,362	3
FIXPI#	1	11	12	0
SIN	0	11	11	0
COS	0	11	11	0
Time (secs)	0.0682	0.8092	0.8774	100
Percent of Total Time	8	92	100	

(Route, 1981)

TABLE 5.30

IBM Singleton's Mixed-Radix Results, N = 504

Instruction Type	Initialize	Number of Times Executed: Transform Factor of 2	4	3	5	Odd	Odd Factor Rotation	Permutation	Total	Percent of Total Time
Float Add	12	4,594	0	4,032	0	4,759	2,783	0	16,180	18
Float Mult.	36	2,894	0	1,344	0	2,604	4,902	0	11,780	38
Float Div.	7	0	0	0	0	2	0	0	9	0
Load Register	104	4,326	0	1,680	0	4,116	1,958	427	12,611	3
Load	124	6,574	3	8,569	0	11,354	6,151	7,226	40,001	15
Store	137	5,290	0	3,865	0	8,462	4,413	4,095	26,262	12
Integer Add	42	1,851	0	1,177	0	4,917	1,958	4,183	14,128	5
Integer Mult.	9	0	0	0	0	0	0	78	87	0
Integer Div.	18	11	0	0	0	3	5	2	39	0
Compare	19	963	3	505	0	1,814	1,249	1,818	6,370	2
Branch	62	972	3	507	0	1,816	1,268	1,981	6,608	2
SIN	13	1	0	0	0	1	5	0	20	1
COS	1	1	0	0	0	1	5	0	8	0
SQRT	1	0	0	0	0	0	0	0	1	0
ATAN	1	0	0	0	0	0	0	0	1	0
Others	38	1,523	0	1,008	0	2,813	699	1,441	7,522	4
Time (secs)	0.0026	0.0521	0	0.0354	0	0.0614	0.0581	0.0174	0.2270	
Percent of Total Time	1	23	0	15	0	27	26	8	100	

(Route, 1981)

TABLE 5.31

IBM Singleton's Mixed-Radix Results, N = 630

Instruction	Initialize	Number of Times Executed								Percent of Total Time
		Transform Factor of					Odd Factor Rotation	Permutation	Total	
		2	4	3	5	Odd				
Float Add	10	2,086	0	5,040	4,033	5,947	4,433	0	21,549	18
Float Mult.	30	1,450	0	1,680	2,020	3,252	7,972	0	16,404	40
Float Div.	6	0	0	0	0	2	0	0	8	0
Load Register	94	1,949	0	2,100	1,260	5,142	2,995	1,193	14,733	3
Load	112	3,139	4	10,711	4,920	14,181	9,978	10,073	53,118	15
Store	121	2,415	0	4,831	4,412	10,568	6,966	5,701	35,014	12
Integer Add	38	899	0	1,471	651	6,138	2,998	5,805	18,000	4
Integer Mult.	9	0	0	0	0	0	0	210	219	0
Integer Div.	19	5	0	0	0	4	6	1	35	0
Compare	18	478	4	631	147	2,263	2,024	2,598	8,162	1
Branch	56	483	4	633	148	2,265	2,048	2,924	8,560	2
SIN	11	1	0	0	0	1	6	0	19	1
COS	1	1	0	0	0	1	6	0	9	0
SQRT	1	0	0	0	0	0	0	0	1	0
ATAN	1	0	0	0	0	0	0	0	1	0
Others	37	635	0	1,260	630	3,516	1,264	1,859	9,201	4
Time (secs)	0.0023	0.0250	0	0.0443	0.0360	0.0767	0.0935	0.0242	0.3020	
Percent of Total Time	1	8	0	15	12	25	31	8	100	

(Route, 1981)

TABLE 5.32

IBM Singleton's Mixed-Radix Results, N = 1008

Instruction Type	Initialize	Number of Times Executed: Transform Factor of 2	4	3	5	Odd	Odd Factor Rotation	Permutation	Total	Percent of Total Time
Float Add	10	0	11,306	8,064	0	9,511	4,769	0	33,660	19
Float Mult.	30	0	6,010	2,688	0	5,196	9,164	0	23,088	39
Float Div.	6	0	0	0	0	2	0	0	8	0
Load Register	94	0	7,585	3,360	0	8,220	2,735	264	22,258	3
Load	112	0	16,896	16,888	0	22,664	11,461	8,732	76,753	15
Store	123	0	15,170	7,480	0	16,892	7,298	4,586	51,549	12
Integer Add	40	0	2,540	2,104	0	9,819	2,746	3,420	20,669	4
Integer Mult.	7	0	0	0	0	0	0	3	10	0
Integer Div.	16	0	9	0	0	3	2	2	32	0
Compare	16	0	784	760	0	3,620	2,311	2,154	9,644	1
Branch	54	0	1,818	762	0	3,622	2,323	2,366	10,944	2
SIN	11	0	7	0	0	1	2	0	21	1
COS	1	0	7	0	0	1	2	0	11	0
SQRT	1	0	0	0	0	0	0	0	1	0
ATAN	1	0	0	0	0	0	0	0	1	0
Others	36	0	2,025	2,016	0	5,621	1,962	1,979	13,639	4
Time (secs)	0.0023	0	0.1150	0.0700	0	0.1224	0.1046	0.0199	0.4342	
Percent of Total Time	1	0	26	16	0	28	24	5	100	

(Route, 1981)

TABLE 5.33

IBM Singleton's Mixed-Radix Results, N = 1260

Instruction Type	Initialize	Number of Times Executed: Transform Factor of 2	4	3	5	Odd	Odd Factor Rotation	Permutation	Total	Percent of Total Time
Float Add	12	7,543	0	10,080	8,065	11,887	9,317	0	46,904	18
Float Mult.	36	4,598	0	3,360	4,036	6,492	16,836	0	35,358	40
Float Div.	7	0	0	0	0	2	0	0	9	0
Load Register	110	7,271	0	4,200	2,520	10,272	6,228	1,019	31,620	3
Load	126	10,510	4	21,421	9,834	28,314	20,997	18,132	109,338	15
Store	137	8,631	0	9,661	8,822	21,101	14,575	10,310	73,237	12
Integer Add	44	3,068	0	2,941	1,302	12,261	6,215	10,621	36,452	4
Integer Mult.	11	0	0	0	0	0	0	198	209	0
Integer Div.	20	12	0	0	0	4	14	2	52	0
Compare	21	1,593	4	1,261	294	4,516	4,260	4,591	16,539	1
Branch	66	1,606	4	1,263	295	4,518	4,308	4,976	17,035	2
SIN	13	3	0	0	0	1	14	0	31	1
COS	1	3	0	0	0	1	14	0	19	0
SQRT	1	0	0	0	0	0	0	0	1	0
ATAN	1	0	0	0	0	0	0	0	1	0
Others	40	2,532	0	2,520	1,260	7,026	2,736	3,584	19,698	4
Time (secs)	0.0027	0.0845	0	0.0886	0.0719	0.1528	0.1974	0.0436	0.6415	
Percent of Total Time	0	13	0	14	11	24	31	7	100	

(Route, 1981)

TABLE 5.34

IBM Singleton's Mixed-Radix Results, N = 2520

Instruction Type	Initialize	Number of Times Executed: Transform Factor of 2	4	3	5	Odd	Odd Factor Rotation	Permutation	Total	Percent of Total Time
Float Add	14	23,050	0	20,160	16,129	23,767	18,086	0	101,206	19
Float Mult.	42	14,638	0	6,720	8,068	12,972	32,746	0	75,186	40
Float Div.	8	0	0	0	0	2	0	0	10	0
Load Register	122	21,586	0	8,400	5,040	20,532	11,953	2,443	70,076	3
Load	140	32,950	4	42,841	19,662	56,574	40,944	37,421	230,536	15
Store	153	26,498	0	19,321	17,642	42,161	28,299	21,436	155,510	12
Integer Add	49	9,175	0	5,881	2,562	24,501	11,914	22,163	76,245	4
Integer Mult.	13	0	0	0	0	0	0	415	428	0
Integer Div.	23	27	0	0	0	4	27	2	83	0
Compare	24	4,759	4	2,521	546	9,016	8,270	9,413	34,552	1
Branch	75	4,792	4	2,523	547	9,018	8,357	10,340	35,655	2
SIN	15	9	0	0	0	1	27	0	52	0
COS	1	9	0	0	0	1	27	0	38	0
SQRT	1	0	0	0	0	0	0	0	1	0
ATAN	1	0	0	0	0	0	0	0	1	0
Others	44	7,587	0	5,040	2,520	14,046	5,473	7,395	42,105	4
Time (secs)	0.0030	0.2622	0	0.1771	0.1437	0.3052	0.3839	0.0903	1.3654	
Percent of Total Time	0	19	0	13	11	22	28	7	100	

(Route, 1981)

TABLE 5.35a

IBM WFTA Results, N = 504

Instruction Type	Number of Times Executed								
	Initialize	Input Additions for Factor of							Nested Mult.
		2	3	5	7	8	9	16	
Float Add	0	0	0	0	2,992	1,764	2,128	0	0
Float Mult.	2,376	0	0	0	0	0	0	0	1,584
Float Div.	0	0	0	0	0	0	0	0	0
Load Register	4,742	0	0	0	1,235	1,012	795	0	10
Load	5,783	3	3	3	4,058	2,810	3,734	2	1,599
Store	5,999	0	0	0	4,320	2,870	3,228	0	1,608
Integer Add	12,208	0	0	1	970	648	745	0	4
Integer Mult.	614	0	0	0	2	3	3	0	0
Integer Div.	22	0	0	0	0	0	0	0	0
Compare	6,432	2	2	1	90	72	65	1	0
Branch	9,819	2	2	2	90	72	65	1	797
Others	254	0	0	0	705	448	567	0	2
Time (secs)	0.0483	0	0	0	0.0176	0.0111	0.0134	0	0.0156
Percent of Total Time	29	0	0	0	10	7	8	0	9

(Route, 1981)

TABLE 5.35b

IBM WFTA Results, N = 504

Instruction Type	Number of Times Executed: Output Additions for Factor of 3	4	5	7	8	9	16	Perm1	Perm2	Total	Percent of Total Time
Float Add	0	0	0	3,344	1,512	2,688	0	0	0	14,428	21
Float Mult.	0	0	0	0	0	0	0	0	0	3,960	17
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	0	0	0	883	767	459	0	81	79	10,063	4
Load	3	3	2	4,938	3,314	5,413	3	4,138	4,138	39,947	19
Store	0	0	0	4,320	2,870	3,732	0	1,443	1,443	31,833	17
Integer Add	0	0	0	970	648	745	1	2,671	2,671	22,282	8
Integer Mult.	0	0	0	2	3	3	0	3	3	636	2
Integer Div.	0	0	0	0	0	0	0	0	0	22	0
Compare	2	2	1	90	72	65	1	575	575	8,048	2
Branch	2	2	1	90	72	65	2	580	580	12,244	6
Others	0	0	0	705	448	567	0	631	631	4,958	4
Time (secs)	0	0	0	0.0189	0.0107	0.0161	0	0.0076	0.0076	0.1669	100
Percent of Total Time	0	0	0	11	6	10	0	5	5	100	

(Route, 1981)

TABLE 5.36a

IBM WFTA Results, N = 630

Instruction Type	Number of Times Executed								
	Initialize	Input Additions for Factor of							Nested Mult.
		2	3	5	7	8	9	16	
Float Add	0	1,260	0	3,168	3,740	0	2,660	0	0
Float Mult.	3,564	0	0	0	0	0	0	0	2,376
Float Div.	0	0	0	0	0	0	0	0	0
Load Register	6,167	1,301	0	1,983	1,547	0	1,023	0	10
Load	10,767	1,808	3	4,166	5,086	2	4,782	2	2,391
Store	9,653	1,419	0	5,349	5,414	0	4,150	0	2,400
Integer Add	16,426	1,342	0	1,585	1,220	0	991	0	4
Integer Mult.	1,016	3	0	2	2	0	3	0	0
Integer Div.	20	0	0	0	0	0	0	0	0
Compare	8,287	357	2	199	116	1	111	1	0
Branch	11,506	357	2	200	116	1	111	1	1,193
Others	1,253	350	0	990	885	0	735	0	2
Time (secs)	0.0724	0.0092	0	0.0211	0.0221	0	0.0170	0	0.0233
Percent of Total Time	28	4	0	8	9	0	7	0	9

(Route, 1981)

TABLE 5.36b

IBM WFTA Results, N = 630

Instruction Type	Number of Times Executed: Output Additions for Factor of 3	4	5	7	8	9	16	Perm1	Perm2	Total	Percent of Total Time
Float Add	0	0	3,564	4,180	0	3,360	0	0	0	21,932	21
Float Mult.	0	0	0	0	0	0	0	0	0	5,940	16
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	0	0	1,587	1,107	0	603	0	333	331	15,992	4
Load	3	3	3,372	6,186	2	6,881	3	7,384	7,384	60,255	18
Store	0	0	3,765	5,414	0	4,780	0	3,303	3,303	48,950	18
Integer Add	0	0	1,584	1,220	0	991	1	3,869	3,869	33,102	8
Integer Mult.	0	0	2	2	0	3	0	3	3	1,039	3
Integer Div.	0	0	0	0	0	0	0	0	0	20	0
Compare	2	2	199	116	1	111	1	985	985	11,476	2
Branch	2	2	199	116	1	111	2	990	990	15,900	5
Others	0	0	990	885	0	735	0	1,261	1,261	9,347	5
Time (secs)	0	0	0.0204	0.0236	0	0.0204	0	0.0136	0.0136	0.2568	100
Percent of Total Time	0	0	8	9	0	8	0	5	5	100	

(Route, 1981)

TABLE 5.37a

IBM WFTA Results, N = 1008

Instruction Type	Number of Times Executed								
	Initialize	Input Additions for Factor of 2	3	5	7	8	9	16	Nested Mult.
Float Add	0	0	0	0	6,732	0	4,788	4,788	0
Float Mult.	5,346	0	0	0	0	0	0	0	3,564
Float Div.	0	0	0	0	0	0	0	0	0
Load Register	13,074	0	0	0	2,775	0	1,775	3,218	10
Load	14,603	3	3	3	9,118	2	8,354	8,291	3,579
Store	15,331	0	0	0	9,710	0	7,218	7,784	3,588
Integer Add	28,550	0	0	1	2,180	0	1,655	3,294	4
Integer Mult.	1,134	0	0	0	2	0	3	3	0
Integer Div.	29	0	0	0	0	0	0	0	0
Compare	16,753	2	2	1	200	1	135	1,080	0
Branch	27,466	2	2	2	200	1	135	1,577	1,787
Others	260	0	0	0	1,585	0	1,267	1,204	2
Time (secs)	0.1138	0	0	0	0.0396	0	0.0300	0.0328	0.0350
Percent of Total Time	29	0	0	0	10	0	8	8	9

(Route, 1981)

TABLE 5.37b

IBM WFTA Results, N = 1008

Instruction Type	Number of Times Executed: Output Additions for Factor of 3	4	5	7	8	9	16	Perm1	Perm2	Total	Percent of Total Time
Float Add	0	0	0	7,524	0	6,048	4,473	0	0	34,353	22
Float Mult.	0	0	0	0	0	0	0	0	0	8,910	16
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	0	0	0	1,983	0	1,019	2,343	81	79	26,357	4
Load	3	3	4	11,098	2	12,133	11,377	7,666	7,666	93,906	19
Store	0	0	0	9,710	0	8,352	7,973	2,451	2,451	74,568	17
Integer Add	0	0	0	2,180	0	1,655	1,279	5,191	5,191	51,180	8
Integer Mult	0	0	0	2	0	3	3	3	3	1,156	2
Integer Div.	0	0	0	0	0	0	0	0	0	29	0
Compare	2	2	1	200	1	135	72	1,079	1,079	20,745	2
Branch	2	2	1	200	1	135	73	1,084	1,084	33,754	7
Others	0	0	0	1,585	0	1,267	1,078	1,135	1,135	10,518	3
Time (secs)	0	0	0	0.0425	0	0.0361	0.0315	0.0141	0.0141	0.3894	100
Percent of Total Time	0	0	0	11	0	9	8	4	4	100	

(Route, 1981)

TABLE 5.38a

IBM WFTA Results, N = 1260

Instruction Type	Number of Times Executed								
	Initialize	Input Additions for Factor of							Nested Mult.
		2	3	5	7	8	9	16	
Float Add	0	0	0	6,336	7,480	0	5,320	0	0
Float Mult.	7,128	0	0	0	0	0	0	0	4,752
Float Div.	0	0	0	0	0	0	0	0	0
Load Register	9,632	0	0	3,963	3,087	0	2,003	0	10
Load	14,825	3	3	8,324	10,146	2	9,402	2	4,767
Store	14,313	0	0	10,695	10,804	0	8,140	0	4,776
Integer Add	29,750	0	0	3,169	2,430	0	1,901	0	4
Integer Mult.	1,647	0	0	2	2	0	3	0	0
Integer Div.	18	0	0	0	0	0	0	0	0
Compare	14,198	2	2	397	226	1	181	1	0
Branch	19,624	2	2	398	226	1	181	1	2,381
Others	1,250	0	0	1,980	1,765	0	1,435	0	2
Time (secs)	0.1258	0	0	0.0421	0.0440	0	0.0337	0	0.0466
Percent of Total Time	26	0	0	9	9	0	7	0	10

(Route, 1981)

TABLE 5.38b

IBM WFTA Results, N = 1260

Instruction Type	Number of Times Executed										Percent of Total Time
	Output Additions for Factor of										
	3	4	5	7	8	9	16	Perm1	Perm2	Total	
Float Add	0	5,040	7,128	8,360	0	6,720	0	0	0	46,384	24
Float Mult.	0	0	0	0	0	0	0	0	0	11,880	17
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	0	3,819	3,171	2,207	0	1,163	0	333	331	29,719	4
Load	3	3,698	6,738	12,346	2	13,601	3	11,794	11,794	107,453	18
Store	0	3,939	7,527	10,804	0	9,400	0	4,563	4,563	89,524	17
Integer Add	0	1,972	3,168	2,430	0	1,901	1	7,019	7,019	60,764	8
Integer Mult.	0	3	2	2	0	3	0	3	3	1,670	2
Integer Div.	0	0	0	0	0	0	0	0	0	18	0
Compare	2	357	397	226	1	181	1	1,615	1,615	19,403	2
Branch	2	357	397	226	1	181	2	1,620	1,620	27,222	4
Others	0	980	1,980	1,765	0	1,435	0	1,891	1,891	16,374	4
Time (secs)	0	0.0259	0.0408	0.0472	0	0.0404	0	0.0217	0.0217	0.4899	100
Percent of Total Time	0	5	8	10	0	8	0	4	4	100	

(Route, 1981)

TABLE 5.39a

IBM WFTA Results, N = 2520

Instruction Type	Number of Times Executed								
	Initialize	Input Additions for Factor of							Nested Mult.
		2	3	5	7	8	9	16	
Float Add	0	0	0	12,672	14,960	8,820	10,640	0	0
Float Mult.	14,256	0	0	0	0	0	0	0	9,504
Float Div.	0	0	0	0	0	0	0	0	0
Load Register	19,715	0	0	7,923	6,167	5,044	3,963	0	10
Load	26,025	3	3	16,640	20,266	14,022	18,642	2	9,519
Store	26,773	0	0	21,387	21,584	14,334	16,120	0	9,528
Integer Add	59,564	0	0	6,337	4,850	3,232	3,721	0	4
Integer Mult.	2,909	0	0	2	2	3	3	0	0
Integer Div.	19	0	0	0	0	0	0	0	0
Compare	29,175	2	2	793	446	356	321	1	0
Branch	42,168	2	2	794	446	356	321	1	4,757
Others	1,250	0	0	3,960	3,525	2,240	2,835	0	2
Time (secs)	0.2448	0	0	0.0842	0.0880	0.0553	0.0669	0	0.0932
Percent of Total Time	24	0	0	8	9	5	7	0	9

(Route, 1981)

TABLE 5.39b

IBM WFTA Results, N = 2520

Instruction Type	Number of Times Executed										Percent of Total Time
	Output Additions for Factor of										
	3	4	5	7	8	9	16	Perm1	Perm2	Total	
Float Add	0	0	14,256	16,720	7,560	13,440	0	0	0	99,068	25
Float Mult.	0	0	0	0	0	0	0	0	0	23,760	16
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	0	0	6,339	4,407	3,823	2,283	0	333	331	60,338	4
Load	3	3	13,470	24,666	16,542	27,041	3	20,614	20,614	228,078	18
Store	0	0	15,051	21,584	14,334	18,640	0	7,083	7,083	193,501	18
Integer Add	0	0	6,336	4,850	3,232	3,721	1	13,319	13,319	122,486	7
Integer Mult.	0	0	2	2	3	3	0	3	3	2,935	2
Integer Div.	0	0	0	0	0	0	0	0	0	19	0
Compare	2	2	793	446	356	321	1	2,875	2,875	38,767	2
Branch	2	2	793	446	356	321	2	2,880	2,880	56,529	4
Others	0	0	3,960	3,525	2,240	2,835	0	3,151	3,151	32,674	4
Time (secs)	0	0	0.0816	0.0944	0.0535	0.0804	0	0.0377	0.0377	1.0177	100
Percent of Total Time	0	0	8	9	5	8	0	4	4	100	

(Route, 1981)

TABLE 5.40

IBM PFA Results, N = 504

Instruction Type	Number of Times Executed											Percent of Total Time
	PFA Control	Short DFT of Factor								Unscrambling	Total	
		2	4	5	7	8	9	16				
Float Add	0	0	0	0	5,184	3,276	4,704	0		0	13,164	36
Float Mult.	0	0	0	0	1,152	252	1,120	0		0	2,524	20
Float Div.	0	0	0	0	0	0	0	0		0	0	0
Load Register	769	0	0	0	648	819	672	0		508	3,416	3
Load	787	0	0	0	4,752	4,032	4,984	0		1,015	15,570	15
Store	1,529	0	0	0	3,672	3,150	3,640	0		1,009	13,000	14
Integer Add	1,683	0	0	0	0	0	0	0		1,199	2,882	2
Integer Mult.	0	0	0	0	0	0	0	0		0	0	0
Integer Div.	3	0	0	0	0	0	0	0		0	3	0
Compare	1,706	0	0	0	0	0	0	0		1,008	2,714	1
Branch	3,218	0	0	0	72	63	56	0		1,008	4,417	4
Others	197	0	0	0	936	1,008	1,008	0		505	3,654	5
Time (secs)	0.0070	0	0	0	0.0322	0.0193	0.0309	0		0.0051	0.0945	100
Percent of Total Time	7	0	0	0	34	21	33	0		5	100	

(Route, 1981)

TABLE 5.41

IBM PFA Results, N = 630

Instruction Type	Number of Times Executed										Percent of Total Time
	PFA Control	Short DFT of Factor							Unscram-bling	Total	
		2	4	5	7	8	9	16			
Float Add	0	1,260	0	4,284	6,480	0	5,880	0	0	17,904	34
Float Mult.	0	0	0	1,260	1,440	0	1,440	0	0	4,100	22
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	2,410	630	0	1,134	810	0	840	0	634	6,458	3
Load	2,119	1,260	0	5,166	5,940	0	6,230	0	1,267	21,982	14
Store	2,542	1,260	0	4,536	4,590	0	4,550	0	1,261	18,739	14
Integer Add	2,924	0	0	0	0	0	0	0	1,861	4,785	3
Integer Mult.	0	0	0	0	0	0	0	0	0	0	0
Integer Div.	4	0	0	0	0	0	0	0	0	4	0
Compare	3,125	0	0	0	0	0	0	0	1,260	4,385	2
Branch	5,330	315	0	126	90	0	70	0	1,260	7,191	4
Others	609	630	0	630	1,170	0	1,260	0	631	4,930	4
Time (secs)	0.0133	0.0074	0	0.0313	0.0402	0	0.0386	0	0.0065	0.1373	100
Percent of Total Time	10	5	0	23	29	0	28	0	5	100	

(Route, 1981)

TABLE 5.42

IBM PFA Results, N = 1008

Instruction Type	Number of Times Executed										Percent of Total Time
	PFA Control	Short DFT of Factor							Unscram-bling	Total	
		2	4	5	7	8	9	16			
Float Add	0	0	0	0	10,368	0	9,408	9,324	0	29,100	37
Float Mult.	0	0	0	0	2,304	0	2,240	1,260	0	5,804	21
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	1,281	0	0	0	1,296	0	1,344	2,016	1,012	6,949	3
Load	1,299	0	0	0	9,504	0	9,968	10,332	2,023	33,126	15
Store	3,041	0	0	0	7,344	0	7,280	8,568	2,017	28,250	14
Integer Add	3,311	0	0	0	0	0	0	0	2,335	5,646	2
Integer Mult.	0	0	0	0	0	0	0	0	0	0	0
Integer Div.	3	0	0	0	0	0	0	0	0	3	0
Compare	3,346	0	0	0	0	0	0	0	2,016	5,362	1
Branch	6,370	0	0	0	144	0	112	63	2,016	8,705	3
Others	325	0	0	0	1,872	0	2,016	2,016	1,009	7,238	4
Time (secs)	0.0135	0	0	0	0.0643	0	0.0617	0.0569	0.0102	0.2066	100
Percent of Total Time	6	0	0	0	31	0	30	28	5	100	

(Route, 1981)

TABLE 5.43

IBM PFA Results, N = 1260

Instruction Type	Number of Times Executed										Percent of Total Time
	PFA Control	Short DFT of Factor							Unscram-bling	Total	
		2	4	5	7	8	9	16			
Float Add	0	0	5,040	8,568	12,960	0	11,760	0	0	38,328	35
Float Mult.	0	0	0	2,520	2,880	0	2,800	0	0	8,200	21
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	3,554	0	1,890	2,268	1,620	0	1,680	0	1,264	12,276	3
Load	3,578	0	5,670	10,332	11,880	0	12,460	0	2,527	46,447	15
Store	5,062	0	5,355	9,072	9,180	0	9,100	0	2,521	40,290	14
Integer Add	5,763	0	0	0	0	0	0	0	3,407	9,170	3
Integer Mult.	0	0	0	0	0	0	0	0	0	0	0
Integer Div.	4	0	0	0	0	0	0	0	0	4	0
Compare	5,931	0	0	0	0	0	0	0	2,520	8,451	1
Branch	10,971	0	315	252	180	0	140	0	2,520	14,378	4
Others	895	0	1,260	1,260	2,340	0	2,520	0	1,261	9,536	4
Time (secs)	0.0249	0	0.0265	0.0626	0.0804	0	0.0771	0	0.0129	0.2844	100
Percent of Total Time	9	0	9	22	28	0	27	0	5	100	

(Route, 1981)

TABLE 5.44

IBM PFA Results, N = 2520

Instruction Type	Number of Times Executed										Percent of Total Time
	PFA Control	Short DFT of Factor							Unscram-bling	Total	
		2	4	5	7	8	9	16			
Float Add	0	0	0	17,136	25,920	16,380	23,520	0	0	82,956	35
Float Mult.	0	0	0	5,040	5,760	1,260	5,600	0	0	17,660	21
Float Div.	0	0	0	0	0	0	0	0	0	0	0
Load Register	5,842	0	0	4,536	3,240	4,095	3,360	0	2,524	23,597	3
Load	5,866	0	0	20,664	23,760	20,160	24,920	0	5,047	100,417	15
Store	10,102	0	0	18,144	18,360	15,750	18,200	0	5,041	85,597	14
Integer Add	11,322	0	0	0	0	0	0	0	6,499	17,821	3
Integer Mult.	0	0	0	0	0	0	0	0	0	0	0
Integer Div.	4	0	0	0	0	0	0	0	0	4	0
Compare	11,543	0	0	0	0	0	0	0	5,040	16,583	1
Branch	21,623	0	0	504	360	315	280	0	5,040	28,122	4
Others	1,467	0	0	2,520	4,680	5,040	5.040	0	2,521	21,268	4
Time (secs)	0.0475	0	0	0.1252	0.0168	0.0965	0.1543	0	0.0257	0.6100	100
Percent of Total Time	8	0	0	21	26	16	25	0	4	100	

(Route, 1981)

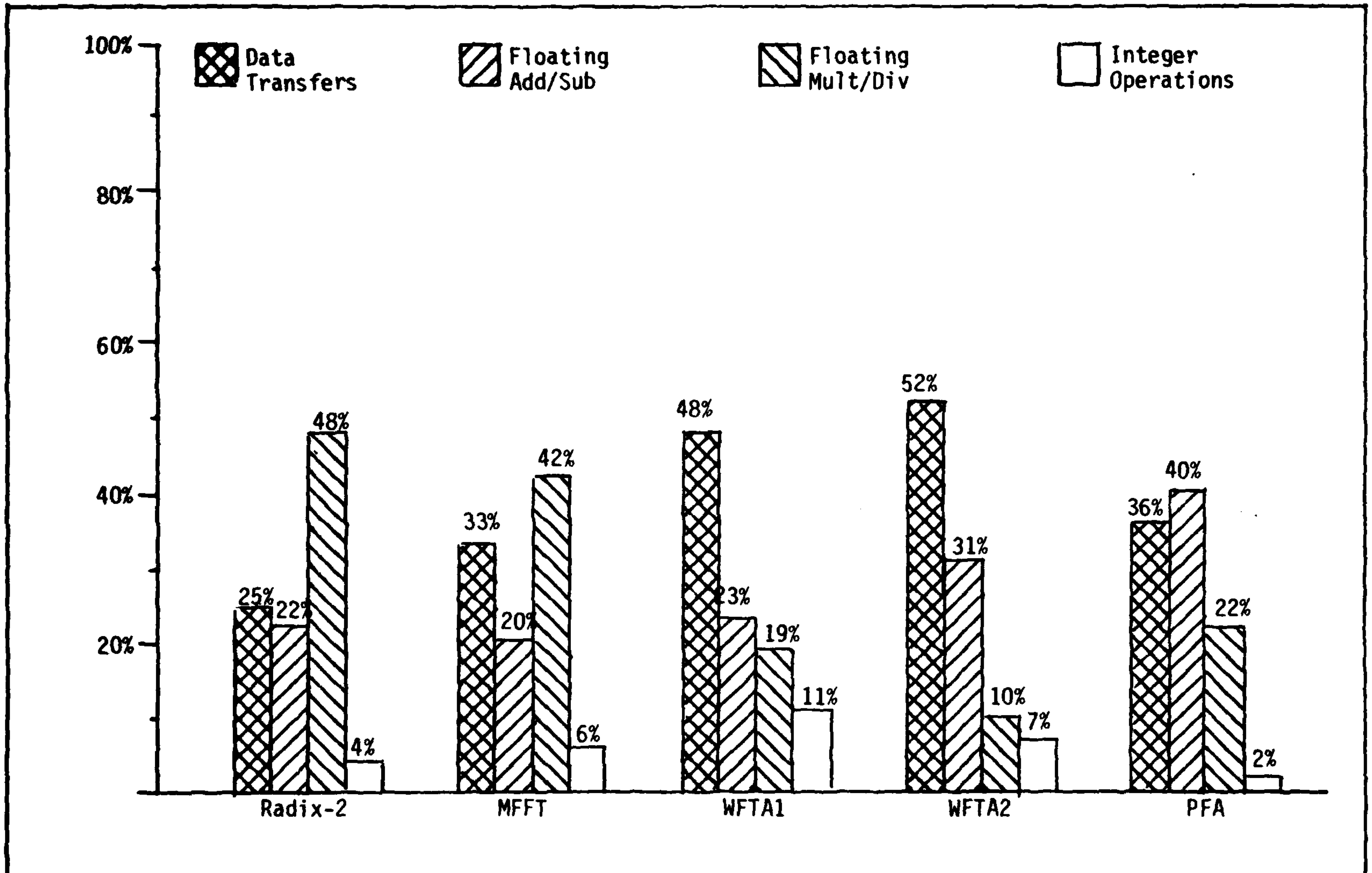


Figure 5.3. Percentage of Time per Instruction Category - IBM 370/155

transfers with 53%, while the radix-2 has the least with 25%.

DEC VAX 11/780

Table 5.45 lists the execution speeds of each of the algorithms and sequence lengths on the VAX 11/780. Using a clock resolution of 16.7 milliseconds and a minimum execution time of 85 miliseconds, the maximum percentage error is 19.6%. The correlation coefficients between the execution speeds and four major instruction categories are:

floating multiply/divide	0.6634
floating add/subtract	0.9495
integer operations	0.9244
data transfers	0.9724.

Tables 5.46 through 5.63 list the number of instructions in each category for each algorithm and sequence length. The correlation coefficients range from 0.6634 for the floating multiplications and divisions to 0.9724 for the data transfers. Thus the floating point accelerator helped reduce the dependence of the execution speed on the floating point operations. However, even though the VAX has an 8 kbyte cache memory, the highest correlation occurred between the data transfers and the execution speed. Since the instruction times are not available for the VAX, the percentage of time spent on each instruction category could not be determined. Attempts were made to determine the instruction timings using the instruction counts and total execution time to form a system of linear equations. But solving this system of equations resulted in some negative instruction times. In order to determine an approximate value for the instruction times, subsets of the system of

TABLE 5.45

Algorithm Execution Speed in Milliseconds for DEC VAX 11/780

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		192	213	133	85
512	183				
630		240	307	207	152
1008		344	523	317	213
1024	360				
1260		521	570	423	308
2048	779				
2520		1127	1280	917	610

TABLE 5.46

VAX Radix-2 Results, N = 512

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Floating Add	0	7423	7423
Floating Sub	0	7423	7423
Floating Mult	0	11260	11260
Floating Div	0	18	18
Integer Add	511	3326	3837
Integer Sub	1017	511	1528
Integer Mult	0	0	0
Integer Div	1014	9	1023
Floating Load	0	0	0
Floating Store	480	5119	5599
Floating Reg Trans	0	0	0
Floating Mem Trans	960	521	1481
Integer Load	481	13834	14315
Integer Store	1026	3364	4390
Integer Reg Trans	0	0	0
Integer Mem Trans	511	1032	1543
Jump	2	0	2
Branch	1525	3847	5372
Add and Branch	511	520	1031
Compare	1014	1032	2046
Convert	0	54	54
Clear	0	9	9
Subroutine Call	1	27	28
Push	2	18	20
Pop	0	0	0
Others	0	2816	2816
Total	9055	62163	71218

TABLE 5.47

VAX Radix-2 Results, N = 1024

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Floating Add	0	16383	16383
Floating Sub	0	16383	16383
Floating Mult	0	24572	24572
Floating Div	0	20	20
Integer Add	1023	7166	8189
Integer Sub	2040	1023	3063
Integer Mult	0	0	0
Integer Div	2037	10	2047
Floating Load	0	0	0
Floating Store	992	11263	12255
Floating Reg Trans	0	0	0
Floating Mem Trans	1984	1034	3018
Integer Load	993	30731	31724
Integer Store	2050	7208	9258
Integer Reg Trans	0	0	0
Integer Mem Trans	1023	2057	3080
Jump	2	0	2
Branch	3060	8200	11260
Add and Branch	1023	1033	2056
Compare	2037	2057	4094
Convert	0	60	60
Clear	0	10	10
Subroutine Call	1	30	31
Push	2	20	22
Pop	0	0	0
Others	0	6144	6144
Total	18267	135404	153671

TABLE 5.48

VAX Radix-2 Results, N = 2048

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Floating Add	0	35839	35839
Floating Sub	0	35839	35839
Floating Mult	0	53244	53244
Floating Div	0	22	22
Integer Add	2047	15358	17405
Integer Sub	4087	2047	6134
Integer Mult	0	0	0
Integer Div	4084	11	4095
Floating Load	0	0	0
Floating Store	1984	24575	26559
Floating Reg Trans	0	0	0
Floating Mem Trans	3968	2059	6027
Integer Load	993	67596	68589
Integer Store	4098	15404	19502
Integer Reg Trans	0	0	0
Integer Mem Trans	2047	4106	6153
Jump	2	0	2
Branch	6131	17417	23548
Add and Branch	2047	2058	4105
Compare	4084	4106	8190
Convert	0	66	66
Clear	0	11	11
Subroutine Call	1	33	34
Push	2	22	24
Pop	0	0	0
Others	0	11265	11265
Total	35575	291078	326653

TABLE 5.49

VAX MFFT Results, N = 504

Instruction Type	Decomposition	Initialization	2	3	4	5	Odd Factor	Rotation	Permutation	Total
F Add	0	0	2295	2688	0	0	3891	1388	0	10262
F Sub	0	0	2296	1344	0	0	867	1392	0	5899
F Mult	0	30	2894	1344	0	0	2604	4902	0	11774
F Div	0	6	0	0	0	0	2	0	0	8
Int Add	30	5	1739	1008	0	0	3315	1260	2093	9450
Int Sub	20	11	932	168	0	0	660	547	1393	3731
Int Mult	8	3	0	0	0	0	0	0	225	236
Int Div	16	1	5	0	0	0	1	5	2	30
F Load	0	0	0	0	0	0	0	0	0	0
F Store	0	0	3024	2016	0	0	1734	1388	0	8162
F Rx	0	3	442	0	0	0	3	0	0	448
F Mx	0	0	247	672	0	0	868	716	2268	4771
Int Load	10	8	4815	3696	0	0	6081	2780	4630	22020
Int St	6	2	1	0	0	0	0	4	10	23
Int Rx	2	0	0	0	0	0	0	0	3	5
Int Mx	20	17	1	0	0	0	724	13	684	1459
Jump	5	4	1	1	0	0	0	8	4	23
Branch	25	9	1473	504	0	0	1815	1256	1582	6664
Add/Br	0	0	0	0	0	0	0	0	0	0
Compare	12	7	1473	504	0	0	1815	1256	1572	6639
Convert	0	44	889	0	0	0	11	25	0	969
Clear	1	2	0	0	0	0	433	0	11	447
Sub Call	1	14	2	0	0	0	2	10	0	29
Push	9	0	0	0	0	0	0	0	0	9
Pop	0	0	0	0	0	0	0	0	0	0
Others	5	2	0	0	0	0	0	0	233	240
Total	170	168	22529	13945	0	0	24826	16950	14710	93298

TABLE 5.50

VAX MFFT Results, N = 630

Instruction Type	Decomposition	Initialization	2	3	4	5	Odd Factor	Rotation	Permutation	Total
F Add	0	0	1042	3360	0	2772	4863	2212	0	14249
F Sub	0	0	1044	1680	0	1135	1083	2216	0	7158
F Mult	0	30	1450	1680	0	2020	3252	7972	0	16404
F Div	0	6	0	0	0	0	2	0	0	8
Int Add	32	5	746	1260	0	630	4503	2252	2533	11961
Int Sub	24	11	317	210	0	63	1219	752	1846	4442
Int Mult	14	3	0	0	0	0	0	0	299	316
Int Div	20	1	7	0	0	0	1	6	1	36
F Load	0	0	0	0	0	0	0	0	0	0
F Store	0	0	1260	2520	0	1260	2166	2516	0	9722
F Rx	0	3	156	0	0	0	3	0	0	162
F Mx	0	0	58	840	0	254	1624	1726	2610	7112
Int Load	11	8	1901	4620	0	1890	7863	5036	5696	27025
Int St	5	2	2	0	0	0	0	4	12	25
Int Rx	2	0	0	0	0	0	0	0	3	5
Int Mx	22	17	2	0	0	0	1264	14	1122	2441
Jump	5	4	4	1	0	1	0	8	4	27
Branch	31	9	590	630	0	189	2347	2026	2049	7871
Add/Br	0	0	0	0	0	0	0	0	0	0
Compare	14	7	590	630	0	189	2347	2026	2037	7840
Convert	0	44	317	0	0	0	11	30	0	402
Clear	1	2	0	0	0	0	541	0	11	555
Sub Call	1	14	2	0	0	0	2	12	0	31
Push	9	0	0	0	0	0	0	0	0	9
Pop	0	0	0	0	0	0	0	0	0	0
Others	5	2	0	0	0	0	0	0	230	237
Total	196	168	9488	17431	0	10403	33091	28808	18453	118038

TABLE 5.51

VAX MFFT Results, N = 1008

Instruction Type	Decomposition	Initialization	2	3	4	5	Odd Factor	Rotation	Permutation	Total
F Add	0	0	0	5376	5530	0	7779	2382	0	21067
F Sub	0	0	0	2688	5781	0	1731	2386	0	12586
F Mult	0	42	0	2688	6010	0	5196	9164	0	23100
F Div	0	8	0	0	0	0	2	0	0	10
Int Add	35	7	0	2016	2280	0	7203	2410	3259	17210
Int Sub	27	11	0	336	264	0	2091	342	466	3537
Int Mult	14	3	0	0	0	0	0	0	3	20
Int Div	24	1	0	0	9	0	1	2	3	40
F Load	0	0	0	0	0	0	0	0	0	0
F Store	0	0	0	4032	2514	0	3462	3920	0	13928
F Rx	0	3	0	0	0	0	3	0	0	6
F Mx	0	0	0	1344	2269	0	2596	2158	2772	11139
Int Load	15	10	0	7392	8078	0	12561	7844	4630	40530
Int St	5	2	0	0	12	0	0	4	6	29
Int Rx	2	0	0	0	2	0	0	0	3	7
Int Mx	25	19	0	0	19	0	2020	10	351	2444
Jump	5	4	0	1	21	0	0	8	1	40
Branch	34	11	0	1008	1808	0	3894	2314	1623	10692
Add/Br	0	0	0	0	0	0	0	0	0	0
Compare	14	9	0	1008	798	0	3894	2314	1623	9660
Convert	0	56	0	0	35	0	11	10	0	112
Clear	1	2	0	0	5	0	865	0	0	873
Sub Call	1	18	0	0	14	0	2	4	0	39
Push	9	0	0	0	0	0	0	0	0	9
Pop	0	0	0	0	0	0	0	0	0	0
Others	7	2	0	0	1010	0	0	0	470	1489
Total	218	208	0	27889	36459	0	53311	35272	15210	168567

TABLE 5.52

VAX MFFT Results, N = 1260

Instruction Type	Decomposition	Initialization	2	3	4	5	Odd Factor	Rotation	Permutation	Total
F Add	0	0	3769	6720	0	5544	9723	3424	0	29180
F Sub	0	0	3771	3360	0	2269	2163	3428	0	14991
F Mult	0	36	4598	3360	0	4036	6492	12086	0	30608
F Div	0	7	0	0	0	0	2	0	0	9
Int Add	24	6	2644	2520	0	1260	9003	3485	5227	24169
Int Sub	24	11	1266	420	0	126	2613	1311	3690	9461
Int Mult	14	3	0	0	0	0	0	0	603	620
Int Div	21	1	13	0	0	0	1	13	2	51
F Load	0	0	0	0	0	0	0	0	0	0
F Store	0	0	5040	5040	0	2520	4326	3476	0	20402
F Rx	0	3	577	0	0	0	3	0	0	583
F Mx	0	0	529	1680	0	506	3244	2594	5814	14367
Int Load	11	9	8201	9240	0	3780	15693	6956	11932	55822
Int St	5	2	2	0	0	0	0	4	11	24
Int Rx	2	0	0	0	0	0	0	0	3	5
Int Mx	23	18	2	0	0	0	2524	21	1778	4366
Jump	5	4	5	1	0	1	0	8	4	28
Branch	31	10	2065	1260	0	378	4866	3072	4027	15709
Add/Br	0	0	0	0	0	0	0	0	0	0
Compare	22	8	2065	1260	0	378	4866	3072	4017	15688
Convert	0	50	1169	0	0	0	11	65	0	1295
Clear	1	2	0	0	0	0	1081	0	11	1095
Sub Call	1	16	6	0	0	0	2	26	0	51
Push	9	0	0	0	0	0	0	0	0	9
Pop	0	0	0	0	0	0	0	0	0	0
Others	5	2	0	0	0	0	0	0	548	555
Total	198	188	35722	34861	0	20798	66613	43041	37667	239088

TABLE 5.53

VAX MFFT Results, N = 2520

Instruction Type	Decomposition	Initialization	2	3	4	5	Odd Factor	Rotation	Permutation	Total
F Add	0	0	11519	13440	0	11088	19443	9028	0	64518
F Sub	0	0	11522	6720	0	4537	4323	9034	0	36136
F Mult	0	42	14638	6720	0	8068	12972	32746	0	75186
F Div	0	8	0	0	0	0	2	0	0	10
Int Add	30	7	7905	5040	0	2520	18003	9142	10442	53089
Int Sub	20	11	3893	840	0	252	5223	2793	7343	20375
Int Mult	8	3	0	0	0	0	0	0	1177	1188
Int Div	16	1	28	0	0	0	1	27	2	75
F Load	0	0	0	0	0	0	0	0	0	0
F Store	0	0	15120	10080	0	5040	8646	10892	0	49778
F Rx	0	3	1834	0	0	0	3	0	0	1840
F Mx	0	0	1165	3360	0	1010	6484	7228	11700	30947
Int Load	10	10	23969	18480	0	7560	31353	21790	23953	127125
Int St	6	2	3	0	0	0	0	6	15	32
Int Rx	2	0	0	0	0	0	0	0	4	6
Int Mx	20	19	3	0	0	0	5044	39	3503	8628
Jump	5	4	6	1	0	1	0	11	4	32
Branch	24	11	6173	2520	0	756	9726	8282	8048	35540
Add/Br	0	0	0	0	0	0	0	0	0	0
Compare	12	9	6173	2520	0	756	9726	8282	8036	35514
Convert	0	56	3713	0	0	0	11	135	0	3915
Clear	1	2	0	0	0	0	2161	0	11	2175
Sub Call	1	18	18	0	0	0	2	54	0	93
Push	9	0	0	0	0	0	0	0	0	9
Pop	0	0	0	0	0	0	0	0	0	0
Others	5	2	0	0	0	0	0	0	1073	1080
Total	169	208	107682	69721	0	41588	133123	119489	75311	547291

TABLE 5.54

VAX WFTA Results, N = 504

Instruction Type	Number of Times Executed							
	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add	0	0	0	3298	0	3972	0	7270
Floating Sub	0	0	0	3586	0	3572	0	7158
Floating Mult	0	2376	0	0	1584	0	0	3960
Floating Div	0	0	0	0	0	0	0	0
Integer Add	6	3908	1079	1929	0	1929	1079	9930
Integer Sub	3	3051	9	6	4	6	9	3088
Integer Mult	6	2073	3	5	0	5	3	2095
Integer Div	0	27	0	0	0	0	0	27
Floating Load	0	0	0	0	0	0	0	0
Floating Store	0	792	0	3424	792	2150	0	7158
Floating Reg Trans	0	0	0	0	0	0	0	0
Floating Mem Trans	0	19	1008	400	0	544	1008	2979
Integer Load	0	4072	2592	6440	1	8616	2592	24313
Integer Store	1	4347	649	250	793	251	649	6940
Integer Reg Trans	0	0	0	540	0	224	0	764
Integer Mem Trans	0	779	578	24	1	402	578	2362
Jump	3	1519	2	2	2	2	2	1532
Branch	3	2303	72	29	1	29	72	2509
Add and Branch	0	2070	575	224	792	224	575	4460
Compare	1	2302	72	29	1	29	72	2506
Convert	0	0	0	0	0	0	0	0
Clear	0	6	0	0	0	0	0	6
Subroutine Call	10	0	0	2	2	2	0	16
Push	21	0	0	1	1	1	0	24
Pop	0	0	0	0	0	0	0	0
Others	3	508	1	4	1	4	1	522
Total	57	30152	6640	20193	3975	21962	6640	89619

TABLE 5.55

VAX WFTA Results, N = 630

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
	Number of Times Executed							
Floating Add	0	0	0	5315	0	6396	0	11711
Floating Sub	0	0	0	4883	0	4708	0	9591
Floating Mult	0	3564	0	0	2376	0	0	5940
Floating Div	0	0	0	0	0	0	0	0
Integer Add	6	5336	1615	3630	0	2993	1615	15195
Integer Sub	3	3798	9	6	4	4	9	3833
Integer Mult	6	2559	3	7	0	6	3	2584
Integer Div	0	20	0	0	0	0	0	20
Floating Load	0	0	0	0	0	0	0	0
Floating Store	0	1188	0	4919	1188	3088	0	10383
Floating Reg Trans	0	0	0	0	0	0	0	0
Floating Mem Trans	0	20	1260	1607	0	1076	1260	5223
Integer Load	0	6337	3506	11210	1	12544	3506	37104
Integer Store	1	7730	1347	1047	1189	679	1347	13340
Integer Reg Trans	0	0	0	675	0	280	0	955
Integer Mem Trans	0	2023	988	93	1	47	988	4140
Jump	3	1899	2	2	2	2	2	1912
Branch	3	3913	356	98	1	53	356	4780
Add and Branch	0	3831	985	744	1188	423	985	8156
Compare	1	3912	356	98	1	53	356	4777
Convert	0	0	0	0	0	0	0	0
Clear	0	6	0	0	0	0	0	6
Subroutine Call	10	0	0	2	2	2	0	16
Push	21	0	0	1	1	1	0	24
Pop	0	0	0	0	0	0	0	0
Others	3	635	1	3	1	2	1	646
Total	57	46771	10428	34340	5955	32357	10428	140336

TABLE 5.56

VAX WFTA Results, N = 1008

Instruction Type	Number of Times Executed							
	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add	0	0	0	7956	0	9315	0	17271
Floating Sub	0	0	0	8352	0	8730	0	17082
Floating Mult	0	5346	0	0	3564	0	0	8910
Floating Div	0	0	0	0	0	0	0	0
Integer Add	6	7925	2087	9359	0	4319	2087	25783
Integer Sub	3	6078	9	6	4	6	9	6115
Integer Mult	6	4095	3	7	0	7	3	4121
Integer Div	0	29	0	0	0	0	0	29
Floating Load	0	0	0	0	0	0	0	0
Floating Store	0	1782	0	7200	1782	4806	0	15570
Floating Reg Trans	0	0	0	0	0	315	0	315
Floating Mem Trans	0	29	2016	1404	0	2106	2016	7571
Integer Load	0	8086	5112	15536	1	19991	5112	53838
Integer Store	1	8371	1153	2194	1783	430	1153	15085
Integer Reg Trans	0	0	0	648	0	819	0	1467
Integer Mem Trans	0	1283	1082	150	1	24	1082	3622
Jump	3	3031	2	2	2	2	2	3044
Branch	3	4321	72	29	1	29	72	4527
Add and Branch	0	4078	1079	1916	1782	404	1079	10338
Compare	1	4320	72	29	1	29	72	4524
Convert	0	0	0	0	0	945	0	945
Clear	0	6	0	0	0	0	0	6
Subroutine Call	10	0	0	2	2	2	0	16
Push	21	0	0	1	1	1	0	24
Pop	0	0	0	0	0	0	0	0
Others	3	1011	1	2018	1	2	1	3037
Total	57	59791	12688	56809	8925	52282	12688	203240

TABLE 5.57

VAX WFTA Results, N = 1260

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
				Number of Times Executed				
Floating Add	0	0	0	10000	0	15312	0	25312
Floating Sub	0	0	0	9136	0	11936	0	21072
Floating Mult	0	7128	0	0	4752	0	0	11880
Floating Div	0	0	0	0	0	0	0	0
Integer Add	6	10302	2875	5941	0	7241	2875	29240
Integer Sub	3	7577	9	4	4	6	9	7612
Integer Mult	6	5075	3	6	0	7	3	5100
Integer Div	0	18	0	0	0	0	0	18
Floating Load	0	0	0	0	0	0	0	0
Floating Store	0	2376	0	9208	2376	8696	0	22656
Floating Reg Trans	0	0	0	0	0	0	0	0
Floating Mem Trans	0	15	2520	2584	0	2152	2520	9791
Integer Load	0	11305	6656	20400	1	29499	6656	74517
Integer Store	1	12694	1977	1259	2377	1661	1977	21946
Integer Reg Trans	0	0	0	720	0	560	0	1280
Integer Mem Trans	0	2651	1618	51	1	93	1618	6032
Jump	3	3787	2	2	2	2	2	3800
Branch	3	6431	356	55	1	98	356	7300
Add and Branch	0	6275	1615	801	2376	1156	1615	13838
Compare	1	6430	356	55	1	98	356	7297
Convert	0	0	0	0	0	0	0	0
Clear	0	5	0	0	0	0	0	5
Subroutine Call	10	0	0	2	2	2	0	16
Push	21	0	0	1	1	1	0	24
Pop	0	0	0	0	0	0	0	0
Others	3	1265	1	2	1	4	1	1277
Total	57	83334	17988	60227	11895	78524	17988	270013

TABLE 5.58

VAX WFTA Results, N = 2520

Instruction Type	Number of Times Executed							
	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add	0	0	0	24410	0	29364	0	53774
Floating Sub	0	0	0	22682	0	22612	0	45294
Floating Mult	0	14256	0	0	9504	0	0	23760
Floating Div	0	0	0	0	0	0	0	0
Integer Add	6	20240	5395	14397	0	14397	5395	59830
Integer Sub	3	15139	9	6	4	6	9	15176
Integer Mult	6	10121	3	7	0	7	3	10147
Integer Div	0	19	0	0	0	0	0	19
Floating Load	0	0	0	0	0	0	0	0
Floating Store	0	4752	0	23456	4752	15502	0	48462
Floating Reg Trans	0	0	0	0	0	0	0	0
Floating Mem Trans	0	19	5040	5168	0	4304	5040	19571
Integer Load	0	21241	12956	49613	1	58909	12956	155676
Integer Store	1	22634	3237	2813	4753	2813	3237	39488
Integer Reg Trans	0	0	0	2700	0	1120	0	3820
Integer Mem Trans	0	3911	2878	93	1	1983	2878	11744
Jump	3	7568	2	2	2	2	2	7581
Branch	3	11472	356	98	1	98	356	12384
Add and Branch	0	11175	2875	1912	4752	1912	2875	25501
Compare	1	11471	356	98	1	98	356	12381
Convert	0	0	0	0	0	0	0	0
Clear	0	6	0	0	0	0	0	6
Subroutine Call	10	0	0	2	2	2	0	16
Push	21	0	0	1	1	1	0	24
Pop	0	0	0	0	0	0	0	0
Others	3	2524	1	4	1	4	1	2538
Total	57	156548	33108	147462	23775	153134	33108	547192

TABLE 5.59

VAX PFA Results, N = 504

Instruction Type	Number of Times Executed: Control	7	8	9	Unscramble	Total
Floating Add	0	2736	1638	2352	0	6726
Floating Sub	0	2448	1638	2352	0	6438
Floating Mult	0	1152	252	1120	0	2524
Floating Div	0	0	0	0	0	0
Integer Add	1515	0	0	0	504	2019
Integer Sub	174	0	0	0	504	678
Integer Mult	0	0	0	0	0	0
Integer Div	3	0	0	0	0	3
Floating Load	0	0	0	0	0	0
Floating Store	191	936	945	191	0	3136
Floating Reg Trans	0	0	0	112	0	112
Floating Mem Trans	0	0	0	0	1008	1008
Integer Load	192	1656	1512	1624	1009	5993
Integer Store	2091	0	0	0	1514	3605
Integer Reg Trans	3	0	0	0	0	3
Integer Mem Trans	1526	504	504	504	2	3040
Jump	2	72	63	112	0	249
Branch	1901	0	0	0	505	2406
Add and Branch	1324	0	0	0	504	1828
Compare	1707	0	0	0	505	2212
Convert	0	0	0	672	0	672
Clear	0	0	0	0	0	0
Subroutine Call	0	0	0	0	0	0
Push	0	0	0	0	0	0
Pop	0	0	0	0	0	0
Others	385	0	0	0	1	386
Total	11014	9504	6552	9912	6056	43038

TABLE 5.60

VAX PFA Results, N = 630

Instruction Type	Number of Times Executed						
	Control	2	5	7	9	Unscramble	Total
Floating Add	0	630	2268	3420	2940	0	9258
Floating Sub	0	630	2016	3060	2940	0	8646
Floating Mult	0	0	1260	1440	1400	0	4100
Floating Div	0	0	0	0	0	0	0
Integer Add	2524	0	0	0	0	630	3154
Integer Sub	406	0	0	0	0	601	1007
Integer Mult	0	0	0	0	0	0	0
Integer Div	4	0	0	0	0	0	4
Floating Load	0	0	0	0	0	0	0
Floating Store	601	945	1134	1170	1330	0	5180
Floating Reg Trans	0	0	0	0	140	0	140
Floating Mem Trans	0	630	0	0	0	1260	1890
Integer Load	602	3150	2016	2070	2030	1261	11129
Integer Store	4331	0	0	0	0	1863	6194
Integer Reg Trans	4	0	0	0	0	0	4
Integer Mem Trans	2537	630	630	630	630	2	5059
Jump	2	0	126	90	140	0	358
Branch	3731	0	0	0	0	631	4362
Add and Branch	1923	0	0	0	0	630	2553
Compare	3126	0	0	0	0	631	3757
Convert	0	0	0	0	840	0	840
Clear	0	0	0	0	0	0	0
Subroutine Call	0	0	0	0	0	0	0
Push	0	0	0	0	0	0	0
Pop	0	0	0	0	0	0	0
Others	1206	0	0	0	0	1	1207
Total	20997	6615	9450	11880	12390	7510	68842

TABLE 5.61

VAX PFA Results, N = 1008

Instruction Type	Number of Times Executed					
	Control	7	9	16	Unscramble	Total
Floating Add	0	5472	4704	4410	0	14586
Floating Sub	0	4896	4704	4914	0	14514
Floating Mult	0	2304	2240	1260	0	5804
Floating Div	0	0	0	0	0	0
Integer Add	3594	0	0	0	1008	4602
Integer Sub	290	0	0	0	886	1176
Integer Mult	0	0	0	0	0	0
Integer Div	3	0	0	0	0	3
Floating Load	0	0	0	0	0	0
Floating Store	886	1872	2128	1953	0	6839
Floating Reg Trans	0	0	224	0	0	224
Floating Mem Trans	0	0	0	0	2016	2016
Integer Load	887	3312	3248	3024	2017	12488
Integer Store	6255	0	0	0	2904	9159
Integer Reg Trans	3	0	0	0	0	3
Integer Mem Trans	3605	1008	1008	1008	2	6631
Jump	2	144	224	63	0	433
Branch	5730	0	0	0	1009	6379
Add and Branch	2708	0	0	0	1008	3716
Compare	4481	0	0	0	1009	5490
Convert	0	0	1344	0	0	1344
Clear	0	0	0	0	0	0
Subroutine Call	0	0	0	0	0	0
Push	0	0	0	0	0	0
Pop	0	0	0	0	0	0
Others	1775	0	0	0	1	1776
Total	29859	19008	19824	16632	11860	97183

TABLE 5.62

VAX PFA Results, N = 1260

Instruction Type	Number of Times Executed						
	Control	4	5	7	9	Unscramble	Total
Floating Add	0	2520	4536	6840	5880	0	19776
Floating Sub	0	2520	4032	6120	5880	0	18552
Floating Mult	0	0	2520	2880	2800	0	8200
Floating Div	0	0	0	0	0	0	0
Integer Add	5044	0	0	0	0	1260	6304
Integer Sub	725	0	0	0	0	887	1612
Integer Mult	0	0	0	0	0	0	0
Integer Div	4	0	0	0	0	0	4
Floating Load	0	0	0	0	0	0	0
Floating Store	887	2205	2268	2340	2660	0	10360
Floating Reg Trans	0	0	0	0	280	0	280
Floating Mem Trans	0	0	0	0	0	2520	2520
Integer Load	888	3465	4032	4140	4060	2521	19106
Integer Store	7709	315	0	0	0	3409	11433
Integer Reg Trans	4	0	0	0	0	0	4
Integer Mem Trans	5057	1260	1260	1260	1260	2	10099
Jump	2	315	252	180	280	0	1029
Branch	6823	0	0	0	0	1261	8084
Add and Branch	4157	0	0	0	0	1260	5417
Compare	5932	0	0	0	0	1261	7193
Convert	0	0	0	0	1680	0	1680
Clear	0	0	0	0	0	0	0
Subroutine Call	0	0	0	0	0	0	0
Push	0	0	0	0	0	0	0
Pop	0	0	0	0	0	0	0
Others	1778	0	0	0	0	1	1779
Total	39010	12600	18900	23760	24780	14382	133432

TABLE 5.63

VAX PFA Results, N = 2520

Instruction Type	Number of Times Executed						
	Control	5	7	8	9	Unscramble	Total
Floating Add	0	9072	13680	8190	11760	0	42702
Floating Sub	0	8064	12240	8190	11760	0	40254
Floating Mult	0	5040	5760	1260	5600	0	17660
Floating Div	0	0	0	0	0	0	0
Integer Add	10084	0	0	0	0	2520	12604
Integer Sub	1244	0	0	0	0	1459	2703
Integer Mult	0	0	0	0	0	0	0
Integer Div	4	0	0	0	0	0	4
Floating Load	0	0	0	0	0	0	0
Floating Store	1459	4536	4680	4725	5320	0	20720
Floating Reg Trans	0	0	0	0	560	0	560
Floating Mem Trans	0	0	0	0	0	5040	5040
Integer Load	1460	8064	8280	7560	8120	5041	38525
Integer Store	14465	0	0	0	0	6501	20966
Integer Reg Trans	4	0	0	0	0	0	4
Integer Mem Trans	10097	2520	2520	2520	2520	2	20179
Jump	2	504	360	315	560	0	1741
Branch	13007	0	0	0	0	2521	15528
Add and Branch	8625	0	0	0	0	2520	11145
Compare	11544	0	0	0	0	2521	14065
Convert	0	0	0	0	3360	0	3360
Clear	0	0	0	0	0	0	0
Subroutine Call	0	0	0	0	0	0	0
Push	0	0	0	0	0	0	0
Pop	0	0	0	0	0	0	0
Others	2922	0	0	0	0	1	2923
Total	74917	37800	47520	32760	49560	28126	270683

equations were set up and solved. Again some of the results were negative and the reasonable results were not sufficiently repeatable between different subsets. The inability to solve the system of equations is due to the fact that the actual system is not linear. The use of instruction overlap, a separate floating point accelerator, and a cache memory introduce nonlinearities into the system. However, the data transfers would be expected to take the greatest percentage of time, given the correlation coefficients and the results of the other architectures.

DEC PDP 11/60

Table 5.64 lists the execution speeds of each of the algorithms and sequence lengths on the PDP 11/60. Using a clock resolution of 16.7 milliseconds and a minimum execution time of 183 milliseconds, the maximum percentage error is 9.1%. The correlation coefficients between the execution speeds and four major instruction categories are:

floating multiply/divide	0.9760
floating add/subtract	0.9846
integer operations	0.9831
data transfers	0.9962.

Tables 5.65 through 5.82 list the instruction counts for each category and sequence length. The correlation coefficients for the PDP 11/60 range from 0.9670 for the floating multiplications and divisions to 0.9962 for the data transfers. The execution speed increase expected from the addition of the floating point processor is limited by the fact that two memory cycles are required to transfer one operand (Digital Equipment Corporation, 1979). In addition, computational overhead is required to set up the addressing for the floating point operand. Figure 5.4 shows that the WFTA2 has the largest percentage of execution time taken by data transfers with 57%, while the MFFT has the smallest percentage with 49%.

TABLE 5.64

Algorithm Execution Speeds in Milliseconds for DEC PDP 11/60

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		295	351	250	183
512	266				
630		411	502	367	261
1008		566	*	*	384
1024	566				
1260		849	*	*	511
2048	1211				
2520		1800	*	*	*

* Unable to execute due to insufficient memory.

TABLE 5.65

DEC PDP 11/60 and PDP 11/50 Radix-2 Results, N = 512

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Floating Add	0	7423	7423
Floating Sub	0	7423	7423
Floating Mult	0	11260	11260
Floating Div	0	9	9
Integer Add	1471	7824	9295
Integer Sub	504	511	1015
Integer Mult	0	0	0
Integer Div	503	520	1023
Load Float	960	21024	21984
Store Float	1440	15394	16834
Load Int	987	3335	4322
Store Int	3060	2833	5893
Reg Trans	480	9727	10207
Mem Trans	512	2853	3365
Branch	2537	2824	5361
Compare	2035	520	2555
Increment	511	520	1031
Decrement	1	2304	2305
Shift	960	9216	10176
Others	745	557	1302
Total	16706	106077	122783

TABLE 5.66

DEC PDP 11/60 and PDP 11/50 Radix-2 Results, N = 1024

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Floating Add	0	16383	16383
Floating Sub	0	16383	16383
Floating Mult	0	24572	24572
Floating Div	0	10	10
Integer Add	3007	30720	33727
Integer Sub	1015	1023	2038
Integer Mult	0	0	0
Integer Div	1014	1033	2047
Load Float	1984	46116	48100
Store Float	2973	33830	36803
Load Int	2010	7176	9186
Store Int	6130	6163	12293
Reg Trans	992	21503	22495
Mem Trans	1024	6185	7209
Branch	5095	6153	11248
Compare	4082	1033	5115
Increment	1023	1033	2056
Decrement	1	5120	5121
Shift	1984	20480	22464
Others	1512	1074	2586
Total	33846	245990	279836

TABLE 5.67

DEC PDP 11/60 and PDP 11/50 Radix-2 Results, N = 2048

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Floating Add	0	35839	35839
Floating Sub	0	35839	35839
Floating Mult	0	53244	53244
Floating Div	0	11	11
Integer Add	6015	67584	73599
Integer Sub	2038	2047	4085
Integer Mult	0	0	0
Integer Div	2037	2058	4095
Load Float	3968	100392	104360
Store Float	5952	73770	79722
Load Int	4025	15369	19394
Store Int	12272	13333	25605
Reg Trans	1984	47003	48987
Mem Trans	2048	13357	15405
Branch	10213	13322	23535
Compare	8177	2058	10235
Increment	2047	2058	4105
Decrement	1	11264	11265
Shift	3968	45056	49024
Others	3031	2103	5134
Total	67776	535707	603483

TABLE 5.68

DEC PDP 11/60 and PDP 11/50 MFFT Results, N = 504

Instruction Type	Initialization	2	3	Odd Factor	Rotation	Permutation	Total
Floating Add/Subtract	5	4591	4032	4758	2780	0	16166
Floating Multiply/Divide	31	2894	1344	2606	4902	0	11777
Integer Operations	41	5602	3192	9129	3193	6501	27658
Data Transfers	241	18443	14784	22678	14226	10957	81329
Total	318	31530	23352	39171	25101	17458	136930

TABLE 5.69

DEC PDP 11/60 and PDP 11/50 MFFT Results, N = 630

Instruction Type	Initialization	2	3	5	Odd Factor	Rotation	Permutation	Total
Floating Add/Subtract	5	2086	5040	4160	5946	4428	0	21665
Floating Multiply/Divide	31	1451	1680	2019	3254	7972	0	16407
Integer Operations	50	2329	3990	1953	12217	5518	8273	34330
Data Transfers	254	8044	18480	11599	30380	24700	12976	106433
Total	340	13910	29190	19731	51797	42618	21249	178835

TABLE 5.70

DEC PDP 11/60 and PDP 11/50 MFFT Results, N = 1008

Instruction Type	Initialization	4	3	Odd Factor	Rotation	Permutation	Total
Floating Add/Subtract	7	9750	8064	9510	4768	0	32099
Floating Multiply/Divide	43	5759	2688	5198	9164	0	22852
Integer Operations	58	9616	6384	19534	6670	5102	47364
Data Transfers	290	44685	29568	49398	29370	9156	162467
Total	398	69810	46704	83640	49972	14258	264782

TABLE 5.71

DEC PDP 11/60 and PDP 11/50 MFFT Results, N = 1260

Instruction Type	Initialization	2	3	5	Odd Factor	Rotation	Permutation	Total
Floating Add/Subtract	6	7450	10080	8318	11886	6852	0	44592
Floating Multiply/Divide	37	4598	3360	4035	6494	12086	0	30610
Integer Operations	51	8960	7980	3806	24052	8270	16380	69499
Data Transfers	265	29809	36960	23191	60730	38119	28181	217255
Total	359	50817	58380	39350	103162	65327	44561	361956

TABLE 5.72

DEC PDP 11/60 and PDP 11/50 MFFT Results, N = 2520

Instruction Type	Initialization	2	3	5	Odd Factor	Rotation	Permutation	Total
Floating Add/Subtract	7	23041	20160	16634	23766	18062	0	101670
Floating Multiply/Divide	43	14638	6730	8067	12974	32746	0	75198
Integer Operations	53	26937	15960	7812	48802	22824	33780	156168
Data Transfers	277	90517	73920	46375	121390	103721	56684	492884
Total	380	155133	116770	78888	206932	177353	90464	825920

TABLE 5.73

DEC PDP 11/60 and PDP 11/50 WFTA Results, N = 504

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add/Subtract	0	0	0	6884	0	7544	0	14428
Floating Multiply/Divide	0	2376	0	0	1584	0	0	3960
Integer Operations	13	7507	2602	5956	2379	6364	2602	27423
Data Transfers	70	14434	5283	26966	6351	26813	5283	85200
Total	83	24317	7885	39806	10314	40721	7885	131011

TABLE 5.74

DEC PDP 11/60 and PDP 11/50 WFTA Results, N = 630

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add/Subtract	0	0	0	10828	0	11104	0	21932
Floating Multiply/Divide	0	3564	0	0	2376	0	0	5940
Integer Operations	13	10426	3516	10604	3567	8420	3516	40062
Data Transfers	70	22817	7360	45161	9519	38698	7360	130985
Total	83	36807	10876	66593	15462	58222	10876	198919

TABLE 5.75

DEC PDP 11/60 and PDP 11/50 WFTA Results, N = 1008

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add/Subtract	0	0	0	16308	0	18045	0	34353
Floating Multiply/Divide	0	5346	0	0	3564	0	0	8910
Integer Operations	13	16061	5122	13488	5349	15185	5122	60340
Data Transfers	70	28993	10323	64988	14271	63935	10323	192903
Total	83	50400	15445	94784	23184	97165	15445	296506

TABLE 5.76

DEC PDP 11/60 and PDP 11/50 WFTA Results, N = 1260

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add/Subtract	0	0	0	19136	0	27248	0	46384
Floating Multiply/Divide	0	7128	0	0	4752	0	0	11880
Integer Operations	13	20430	6666	17986	7131	21890	6666	80782
Data Transfers	70	40630	13695	77294	19023	96568	13695	260975
Total	83	68188	20361	114416	30906	145706	20361	400021

TABLE 5.77

DEC PDP 11/60 and PDP 11/50 WFTA Results, N = 2520

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Floating Add/Subtract	0	0	0	47092	0	51976	0	99068
Floating Multiply/Divide	0	14256	0	0	9504	0	0	23760
Integer Operations	13	40451	12966	43206	14259	45246	12966	169107
Data Transfers	70	76329	26295	193292	18031	190944	26295	531256
Total	83	131036	39261	283590	41794	288166	39261	823191

TABLE 5.78

DEC PDP 11/60 and PDP 11/50 PFA Results, N = 504

Instruction Type	Control	7	8	9	Unscramble	Total
Floating Add	0	2736	1638	2352	0	6726
Floating Sub	0	2448	1638	2352	0	6438
Floating Mult	0	1152	252	1120	0	2524
Floating Div	0	0	0	0	0	0
Integer Add	1515	1872	2016	2016	2520	9939
Integer Sub	169	0	0	0	504	673
Integer Mult	0	0	0	0	0	0
Integer Div	6	0	0	0	0	6
Load Float	0	5256	4599	4928	1008	15791
Store Float	0	3888	3213	3696	1008	11805
Load Int	391	792	1008	1120	506	3817
Store Int	1714	0	0	0	1	1715
Reg Trans	1321	792	1008	896	1512	5529
Mem Trans	2305	72	63	56	505	3001
Branch	2836	0	0	0	1008	3844
Compare	2645	0	0	0	1008	3653
Increment	1324	0	0	0	504	1828
Decrement	191	0	0	0	0	191
Others	1718	2016	2142	2352	2017	10245
Total	16135	21024	17577	20888	12101	87725

TABLE 5.79

DEC PDP 11/60 and PDP 11/50 PFA Results, N = 630

Instruction Type	Control	2	5	7	9	Unscramble	Total
Floating Add	0	630	2268	3420	2940	0	9258
Floating Sub	0	630	2016	3060	2940	0	8646
Floating Mult	0	0	1260	1440	1400	0	4100
Floating Div	0	0	0	0	0	0	0
Integer Add	2524	1260	1572	2340	2520	3150	13306
Integer Sub	401	0	0	0	0	601	1002
Integer Mult	0	0	0	0	0	0	0
Integer Div	8	0	0	0	0	0	8
Float Load	0	1890	6930	6570	6160	1260	22810
Float Store	0	1575	3906	4860	4620	1260	16221
Int Load	1213	630	1260	990	1400	632	6125
Int Store	3134	0	378	0	0	1	3513
Reg Trans	1919	630	630	990	1120	1890	7179
Mem Trans	4958	315	126	90	70	631	6190
Branch	4443	0	0	0	0	1260	5703
Compare	3842	0	0	0	0	1260	5102
Increment	1923	0	0	0	0	630	2553
Decrement	601	0	0	0	0	0	601
Others	3140	1890	2016	2520	3220	2521	15307
Total	28106	9450	22302	26280	26390	15096	127624

TABLE 5.80

DEC PDP 11/60 and PDP 11/50 PFA Results, N = 1008

Instruction Type	Control	7	9	16	Unscramble	Total
Floating Add	0	5472	4704	4410	0	14586
Floating Sub	0	4896	4704	4914	0	14514
Floating Mult	0	2304	2240	1260	0	5804
Floating Div	0	0	0	0	0	0
Integer Add	3594	3744	4032	4158	5040	20568
Integer Sub	285	0	0	0	886	1171
Integer Mult	0	0	0	0	0	0
Integer Div	6	0	0	0	0	6
Float Load	0	10512	9856	11592	2016	33976
Float Store	0	7776	7392	9261	2016	26445
Int Load	1781	1584	2240	2142	1010	8757
Int Store	4488	0	0	0	1	4489
Reg Trans	2705	1584	1792	2016	3024	11121
Mem Trans	7164	144	112	63	1009	8492
Branch	6299	0	0	0	2016	8315
Compare	5413	0	0	0	2016	7429
Increment	2708	0	0	0	1008	3716
Decrement	886	0	0	0	0	886
Others	4492	4032	5152	4347	4033	22056
Total	39821	42048	42224	44163	24075	192331

TABLE 5.81

DEC PDP 11/60 and PDP 11/50 PFA Results, N = 1260

Instruction Type	Control	4	5	7	9	Unscramble	Total
Floating Add	0	2520	4536	6840	5880	0	19776
Floating Sub	0	2520	4032	6120	5880	0	18552
Floating Mult	0	0	2520	2880	2800	0	8200
Floating Div	0	0	0	0	0	0	0
Integer Add	5044	2520	3024	4680	5040	6300	26608
Integer Sub	720	0	0	0	0	887	1607
Integer Mult	0	0	0	0	0	0	0
Integer Div	8	0	0	0	0	0	8
Float Load	0	7245	13860	13140	12320	2520	49085
Float Store	0	5040	7812	9720	9240	2520	34332
Int Load	1785	2205	2520	1980	2800	1262	12552
Int Store	5940	945	756	0	0	1	7642
Reg Trans	4153	1260	1260	1980	2240	3780	14673
Mem Trans	8622	315	252	180	140	1261	10770
Branch	9197	0	0	0	0	2520	11717
Compare	8310	0	0	0	0	2520	10830
Increment	4157	0	0	0	0	1260	5417
Decrement	887	0	0	0	0	0	887
Shift	4157	2520	3528	4680	5600	5040	25525
Others	1789	630	504	360	840	1	4124
Total	54769	27720	44604	52560	52780	29872	262305

TABLE 5.82

DEC PDP 11/60 and PDP 11/50 PFA Results, N = 2520

Instruction Type	Control	5	7	8	9	Unscramble	Total
Floating Add	0	9072	13680	8190	11760	0	42702
Floating Sub	0	8064	12240	8190	11760	0	40254
Floating Mult	0	5040	5760	1260	5600	0	17660
Floating Div	0	0	0	0	0	0	0
Integer Add	10084	6048	9360	10080	10080	12600	58252
Integer Sub	1239	0	0	0	0	1459	2698
Integer Mult	0	0	0	0	0	0	0
Integer Div	8	0	0	0	0	0	8
Float Load	0	27720	26280	22995	24640	5040	106675
Float Store	0	15624	19440	16065	18480	5040	74649
Int Load	2929	5040	3960	5040	5600	2522	25091
Int Store	11552	1512	0	0	0	1	13065
Reg Trans	8621	2520	3960	5040	4480	7560	32181
Mem Trans	15950	504	360	315	280	2521	19930
Branch	18705	0	0	0	0	5040	23745
Compare	17246	0	0	0	0	5040	22286
Increment	8625	0	0	0	0	2520	11145
Decrement	1459	0	0	0	0	0	1459
Shift	8625	7056	9360	10080	11200	10080	56401
Others	2933	1008	720	630	1680	1	6972
Total	107976	89208	105120	87885	105560	59424	555173

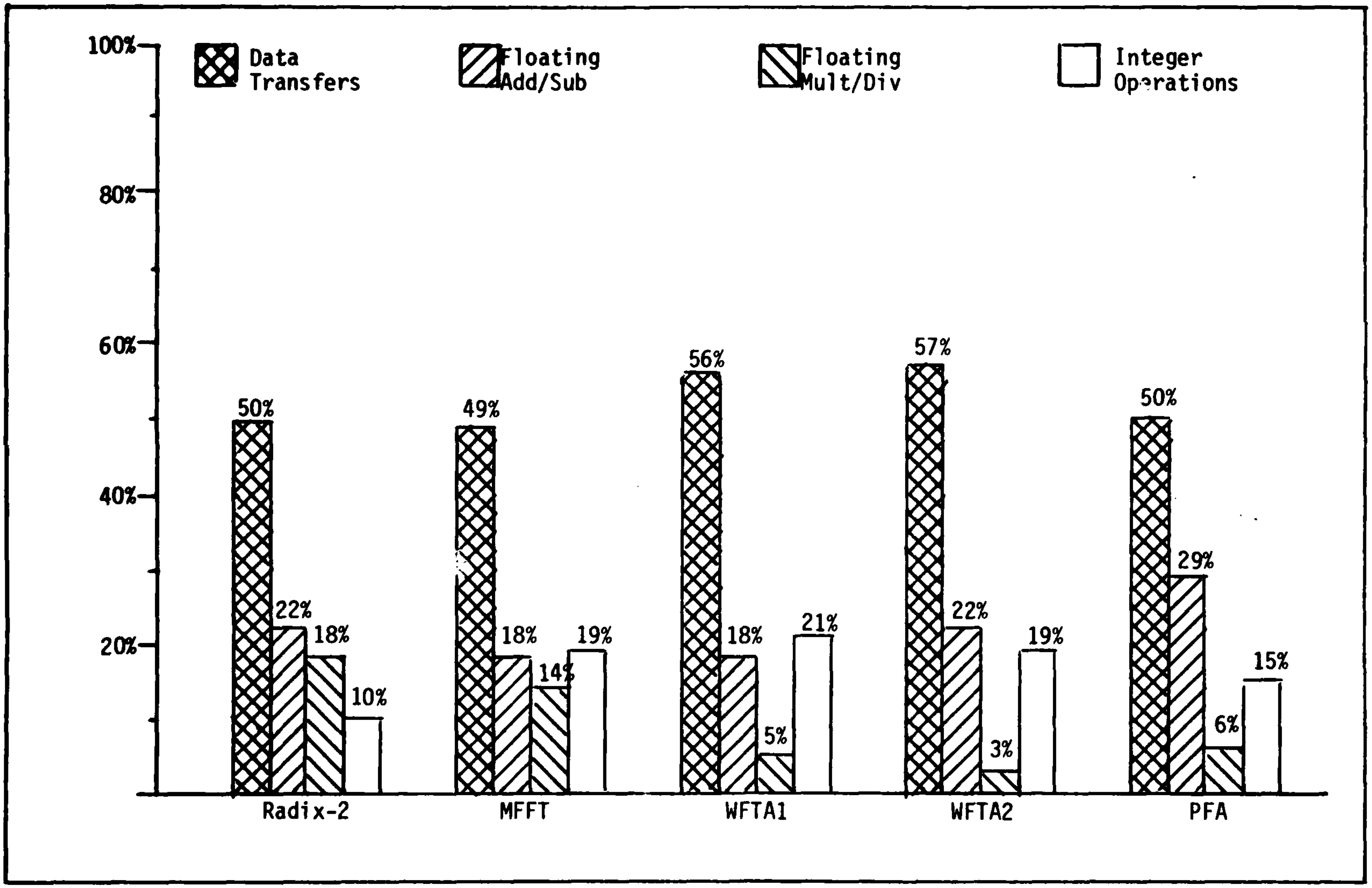


Figure 5.4. Percentage of Time per Instruction Category - DEC PDP 11/60

DEC PDP 11/50

Table 5.83 lists the execution speeds of each of the algorithms and sequence lengths on the PDP 11/50. Using a clock resolution of 16.7 milliseconds and a minimum execution time of 411 milliseconds, the maximum percentage error is 4.1%. The correlation coefficients between the execution speeds and four major instruction categories are:

floating multiply/divide	0.9186
floating add/subtract	0.9575
integer operations	0.9750
data transfers	0.9966.

Tables 5.65 through 5.82, which are the same tables as those for the PDP 11/60, list the instruction counts for each category and sequence length. The values of the correlation coefficients for the PDP 11/50 range from 0.9186 for the floating multiplications and divisions to 0.9966 for the data transfers. Thus the execution speed is most closely related to the number of data transfers. The reason for the importance of the data transfers can be seen in the instruction timings. A floating addition requires 5.66 microseconds, and a floating multiply requires 7.61 microseconds, while a floating load requires 4.20 microseconds (Digital Equipment Corporation, 1976). Thus, if every floating operation requires two loads, then the time taken by the floating loads is more than the time taken by the floating operations. As shown in Figure 5.5, the radix-2 has the greatest percentage of execution time taken

TABLE 5.83

Algorithm Execution Speeds in Milliseconds for DEC PDP 11/50

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		793	771	551	411
512	678				
630		1054	1120	835	602
1008		1543	1834	1291	952
1024	1452				
1260		2250	*	*	1240
2048	3128				
2520		4780	*	*	2651

* Unable to execute due to insufficient memory.

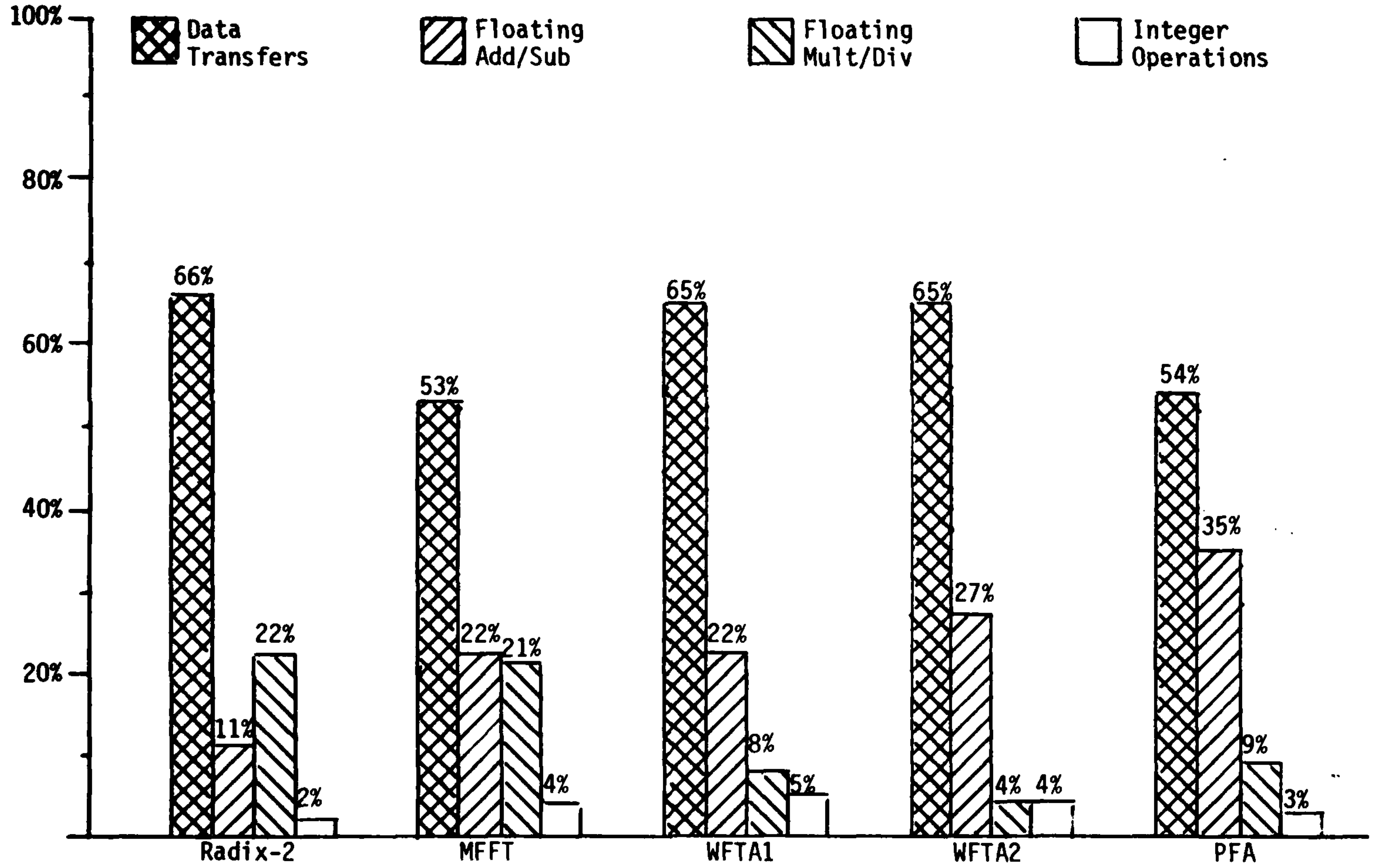


Figure 5.5. Percentage of Time per Instruction Category - DEC PDP 11/50

by data transfers with 66%, while the MFFT has the smallest percentage with 53%.

Cromemco Z-2D

Table 5.84 lists the execution speeds of each of the algorithms and sequence lengths on the Cromemco Z-2D. Using a timing accuracy of 1 second and a minimum execution time of 8 seconds, the maximum percentage error is 12.5%. The correlation coefficients between the execution speeds and four major instruction categories are:

floating multiply/divide	0.9957
floating add/subtract	0.9668
integer operations	0.9954
data transfers	0.9919.

Tables 5.85 through 5.102 list the instruction counts for each of the algorithms and sequence lengths. The correlation coefficients for the Cromemco Z-2D range from 0.9668 for the floating additions and subtractions to 0.9957 for the floating multiplications and divisions. Thus, the execution speed is most closely related to the number of floating point multiplications and divisions. This is because the floating multiplications and divisions must be done with software routines, which are slower than floating point hardware. Thus, as expected, the PFA is the fastest since it has the fewest floating point multiplications, while the MFFT is the slowest since it has the most floating point multiplications. Figure 5.6 shows the percentage of time taken by each of the instruction categories. The radix-2 has the greatest percentage of time taken by floating multiplications with 63%, while the WFTA2 has the smallest with 20%. The WFTA could not be executed due to

TABLE 5.84

Algorithm Execution Speeds in Milliseconds for Cromemco Z-2D

Length	Radix-2	MFFT	WFTA1	WFTA2	PFA
504		21000	*	*	8000
512	18000				
630		29000	*	*	12000
1008		43000	*	*	15000
1024	40000				
1260		62000	*	*	23000
2048	97000				
2520		135000	*	*	*

* Unable to execute due to insufficient memory.

TABLE 5.85

Cromemco Z-2D Radix-2 Results, N = 512

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Real Add	0	7423	7423
Real Subtract	0	7423	7423
Real Multiply	0	11260	11260
Real Divide	0	9	9
Integer Add	3631	46080	49711
Integer Subtract	6598	5648	12246
Integer Multiply	0	0	0
Integer Divide	503	9	512
Load	13378	113563	126941
Store	5507	18970	24477
Register Transfer	16393	36667	53060
Increment	512	529	1041
Decrement	1	0	1
Branch	2035	2824	4859
Jump	502	0	502
Integer Exponentiation	1	9	10
Shift	3048	0	3048
Logical	6096	0	6096
Others	4404	42067	46471
Total	62609	292481	355090

TABLE 5.86

Cromemco Z-2D Radix-2 Results, N = 1024

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Real Add	0	16383	16383
Real Subtract	0	16383	16383
Real Multiply	0	24572	24572
Real Divide	0	10	10
Integer Add	7471	102400	109871
Integer Subtract	13249	12306	25555
Integer Multiply	0	0	0
Integer Divide	1014	10	1024
Load	27196	250030	277226
Store	11136	42013	53149
Register Transfer	33021	80962	113983
Increment	1024	1043	2067
Decrement	1	0	1
Branch	4082	6153	10235
Jump	1013	0	1013
Integer Exponentiation	1	10	11
Shift	6118	0	6118
Logical	12236	0	12236
Others	9011	92253	101264
Total	126573	644528	771101

TABLE 5.87

Cromemco Z-2D Radix-2 Results, N = 2048

Instruction Type	Number of Times Executed		
	Bit-Reversal	Butterfly	Total
Real Add	0	35839	35839
Real Subtract	0	35839	35839
Real Multiply	0	53244	53244
Real Divide	0	11	11
Integer Add	14943	225280	240223
Integer Subtract	26556	26644	53200
Integer Multiply	0	0	0
Integer Divide	2037	11	2048
Load	54454	545985	600439
Store	22301	92192	114493
Register Transfer	66177	177225	243402
Increment	2048	2069	4117
Decrement	1	0	1
Branch	8177	13322	21499
Jump	2036	0	2036
Integer Exponentiation	1	11	12
Shift	12260	0	12260
Logical	24520	0	24520
Others	18034	200807	218841
Total	253545	1408479	1662024

TABLE 5.88

Cromemco Z-2d MFFT Results, N = 504

Instruction Type	Initialization	2	3	Odd Factor	Rotation	Permutation	Total
Real Addition/Subtraction	0	4591	4032	4758	2780	0	16161
Real Multiplication/Division	31	2894	1344	2606	4902	0	11777
Integer Operations	105	16289	9912	23500	9148	16148	75102
Data Transfers	526	61760	35616	70762	40467	48455	257586
Total	662	85534	50904	101626	57297	64603	360626

TABLE 5.89

Cromemco Z-2D MFFT Results, N = 630

Instruction Type	Initialization	2	3	5	Odd Factor	Rotation	Permutation	Total
Real Addition/Subtraction	0	2086	5040	3907	7146	4428	0	22607
Real Multiply/Division	31	1450	1680	2018	3254	7972	0	16405
Integer Operations	103	6650	12390	5481	31171	15830	20302	91927
Data Transfers	564	26145	44520	25841	95679	67810	61524	322083
Total	698	36331	63630	37247	137250	96040	81826	453022

TABLE 5.90

Cromemco Z-2D MFFT Results, N = 1008

Instruction Type	Initialization	4	3	Odd Factor	Rotation	Permutation	Total
Real Addition/Subtraction	0	11299	8064	9510	4768	0	33641
Real Multiplication/Division	43	5508	2688	5198	9164	0	22601
Integer Operations	139	24781	19824	50257	21080	13791	129872
Data Transfers	688	95787	71232	154948	79360	45061	447076
Total	870	137375	101808	219913	114372	58852	633190

TABLE 5.91

Cromemco Z-2D MFFT Results, N = 1260

Instruction Type	Initialization	2	3	5	Odd Factor	Rotation	Permutation	Total
Real Addition/Subtraction	0	7540	10080	7813	11886	6852	0	44171
Real Multiply/Division	37	4598	3360	4034	6494	12086	0	30609
Integer Operations	119	25885	24780	10962	62803	23053	41899	189501
Data Transfers	604	96145	89040	49907	193630	102434	125053	656813
Total	760	134168	127260	72716	274813	144425	166952	921094

TABLE 5.92

Cromemco Z-2D MFFT Results, N = 2520

Instruction Type	Initialization	2	3	5	Odd Factor	Rotation	Permutation	Total
Real Addition/Subtraction	0	23041	20160	15625	23766	18062	0	100654
Real Multiply/Division	43	14638	6720	8066	12974	32746	0	75187
Integer Operations	125	76521	49560	21924	125533	66519	83897	424079
Data Transfers	584	292644	178080	103331	387040	279094	249795	1490568
Total	752	406844	254520	148946	549313	396421	333692	2090488

TABLE 5.93

Cromemco Z-2D WFTA Results, N = 504

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Real Add/Subtract	0	0	0	6884	0	7544	0	14428
Real Multiply/Divide	0	2376	0	0	1584	0	0	3960
Integer Operations	9	29152	6700	15415	7128	15779	6700	80883
Data Transfers	72	85003	15981	50186	21389	50167	15981	238779
Total	81	116531	22681	72485	30101	73490	22681	338050

TABLE 5.94

Cromemco Z-2D WFTA Results, N = 630

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Real Add/Subtract	0	0	0	10198	0	11104	0	21302
Real Multiply/Divide	0	3564	0	0	2376	0	0	5940
Integer Operations	9	41025	8906	27018	10692	23704	8906	120260
Data Transfers	72	117222	22758	86029	32081	73518	22758	354438
Total	81	161811	31664	123245	45149	108326	31664	501940

TABLE 5.95

Cromemco Z-2D WFTA Results, N = 1008

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Real Add/Subtract	0	0	0	16245	0	18045	0	34290
Real Multiply/Divide	0	5346	0	0	3564	0	0	8910
Integer Operations	9	61161	13252	52810	16038	34863	13252	191385
Data Transfers	72	171627	31920	142343	48119	114428	31920	540429
Total	81	238134	45172	211398	67721	167336	45172	775014

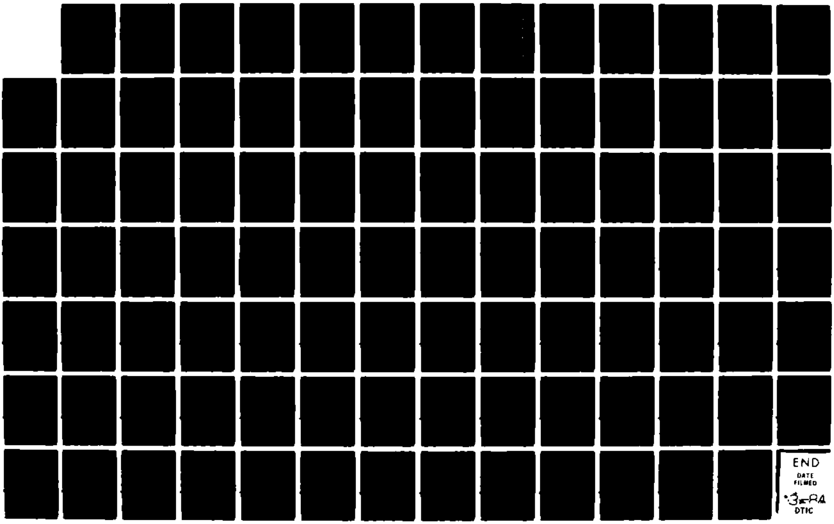
TABLE 5.96

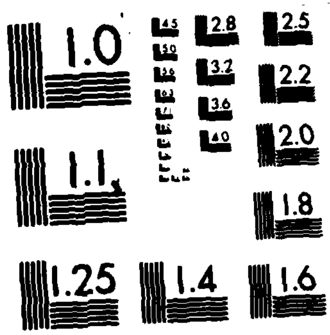
Cromemco Z-2D WFTA Results, N = 1260

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Real Add/Subtract	0	0	0	19136	0	27248	0	46384
Real Multiply/Divide	0	7128	0	0	4752	0	0	11880
Integer Operations	9	80092	17096	48212	21384	57777	17096	241666
Data Transfers	72	225169	42288	153665	64157	183169	42288	710808
Total	81	312389	59384	221013	90293	268194	59384	1010738

AD-A138 465 EFFECTS OF COMPUTER ARCHITECTURE ON FFT (FAST FOURIER TRANSFORM) ALGORITH..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. M A MEHALIC 3/3

UNCLASSIFIED DEC 83 AFIT/GE/EE/83D-47 F/G 12/1 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

TABLE 5.97

Cromemco Z-2D WFTA Results, N = 2520

Instruction Type	Driver	INISHL	PERM1	WEAVE1	MULT	WEAVE2	PERM2	Total
Real Add/Subtract	0	0	0	47092	0	51976	0	99068
Real Multiply/Divide	0	14256	0	0	9504	0	0	23760
Integer Operations	9	158309	33476	116563	42768	114409	33476	499010
Data Transfers	72	440495	81348	373397	128309	358245	81348	1463214
Total	81	613060	114824	537052	180581	524630	114824	2085052

TABLE 5.98

Cromemco Z-2D PFA Results, N = 504

Instruction Type	Control	7	8	9	Unscramble	Total
Floating Add	0	3024	1638	2464	0	7126
Floating Subtract	0	2160	1638	2240	0	6038
Floating Multiply	0	1152	252	1120	0	2524
Floating Divide	0	0	0	0	0	0
Integer Add	4160	4032	3969	3976	5544	21681
Integer Subtract	8644	0	0	0	4032	12676
Integer Multiply	0	0	0	0	0	0
Integer Divide	3	0	0	0	0	3
Load	16950	19656	15246	18592	13610	84054
Store	8673	3024	2961	2968	4033	21659
Register Transfer	19336	2520	2457	2464	12096	38873
Increment	4331	0	0	0	2520	6851
Decrement	1327	0	0	0	504	1831
Branch	2836	0	0	0	1008	3844
Jump	0	72	63	56	0	191
Integer Exponentiation	0	0	0	0	0	0
Shift	1321	0	0	0	504	1825
Logical	5284	0	0	0	2016	7300
Others	191	8784	7056	8400	2017	26448
Total	73056	44424	35280	42280	47884	242924

TABLE 5.99

Cromemco Z-2D PFA Results, N = 630

Instruction Type	Control	2	5	7	9	Unscramble	Total
Floating Add	0	630	2268	3780	3080	0	9758
Floating Subtract	0	630	2016	2700	2800	0	8146
Floating Multiply	0	0	1260	1440	1400	0	4100
Floating Divide	0	0	0	0	0	0	0
Integer Add	6366	4725	5040	5040	4970	6930	33071
Integer Subtract	13516	0	0	0	0	4982	18498
Integer Multiply	0	0	0	0	0	0	0
Integer Divide	4	0	0	0	0	0	4
Load	28155	10000	20664	24570	23240	16896	123605
Store	13725	2520	3780	3780	3710	5012	32527
Register Transfer	30965	2835	3150	3150	3080	14946	58126
Increment	6770	0	0	0	0	3121	9891
Decrement	1927	0	0	0	0	630	2557
Branch	4443	0	0	0	0	1260	5703
Jump	0	315	126	90	70	0	601
Integer Exponentiation	0	0	0	0	0	0	0
Shift	1919	0	0	0	0	630	2549
Logical	7676	0	0	0	0	2520	10196
Others	601	4410	9576	10980	10500	2521	38588
Total	116067	26145	47880	55530	52850	59448	357920

TABLE 5.100

: Cromemco Z-2D PFA Results, N = 1008

Instruction Type	Control	7	9	16	Unscramble	Total
Floating Add	0	6048	4928	4410	0	15386
Floating Subtract	0	4320	4480	4914	0	13714
Floating Multiply	0	2304	2240	1260	0	5804
Floating Divide	0	0	0	0	0	0
Integer Add	9007	8064	7952	8001	11088	44112
Integer Subtract	18570	0	0	0	7820	26390
Integer Multiply	0	0	0	0	0	0
Integer Divide	3	0	0	0	0	3
Load	38893	39312	37184	39690	26730	181809
Store	19178	6048	5936	5985	7943	45090
Register Transfer	42194	5040	4928	4977	23460	80599
Increment	9294	0	0	0	4918	14212
Decrement	2711	0	0	0	1008	3719
Branch	6299	0	0	0	2016	8315
Jump	0	144	112	0	0	256
Integer Exponentiation	0	0	0	0	0	0
Shift	2705	0	0	0	1008	3713
Logical	10820	0	0	0	4032	14852
Others	886	17568	16800	19908	4033	59195
Total	160560	88848	84560	89145	94056	517169

TABLE 5.101

Cromemco Z-2D PFA Results, N = 1260

Instruction Type	Control	4	5	7	9	Unscramble	Total
Floating Add	0	2520	4536	7560	6160	0	20776
Floating Subtract	0	2520	4032	5400	5600	0	17552
Floating Multiply	0	0	2520	2880	2800	0	8200
Floating Divide	0	0	0	0	0	0	0
Integer Add	13354	9765	10080	10080	9940	13860	67079
Integer Subtract	28130	0	0	0	0	9334	37464
Integer Multiply	0	0	0	0	0	0	0
Integer Divide	4	0	0	0	0	0	4
Load	56579	27720	41328	49140	46480	32530	253777
Store	28306	7245	7560	7560	7420	9708	67799
Register Transfer	63637	5985	6300	6300	6160	28002	116384
Increment	14077	0	0	0	0	5927	20004
Decrement	4161	0	0	0	0	1260	5421
Branch	9197	0	0	0	0	2520	11717
Jump	0	315	252	180	140	0	887
Integer Exponentiation	0	0	0	0	0	0	0
Shift	4153	0	0	0	0	1260	5413
Logical	16612	0	0	0	0	5040	21652
Others	887	11340	19152	21960	21000	5041	79380
Total	239097	67410	95760	111060	105700	114482	733509

TABLE 5.102

Cromemco Z-2D PFA Results, N = 2520

Instruction Type	Control	5	7	8	9	Unscramble	Total
Floating Add	0	9072	15120	8190	12320	0	44702
Floating Subtract	0	8064	10800	8190	11200	0	38254
Floating Multiply	0	5040	5760	1260	5600	0	17660
Floating Divide	0	0	0	0	0	0	0
Integer Add	27330	20160	20160	19845	19880	27720	135095
Integer Subtract	57120	0	0	0	0	18038	75158
Integer Multiply	0	0	0	0	0	0	0
Integer Divide	4	0	0	0	0	0	4
Load	112951	82656	98280	76230	92960	63798	526875
Store	57349	15120	15120	14805	14840	19100	136334
Register Transfer	128267	12600	12600	12285	12320	54114	232186
Increment	28572	0	0	0	0	11539	40111
Decrement	8629	0	0	0	0	2520	11149
Branch	18705	0	0	0	0	5040	23745
Jump	0	504	360	315	280	0	1459
Integer Exponentiation	0	0	0	0	0	0	0
Shift	8621	0	0	0	0	2520	11141
Logical	34484	0	0	0	0	10080	44564
Others	1459	38304	43920	35280	42000	10081	171044
Total	483491	191520	222120	176400	211400	224550	1509481

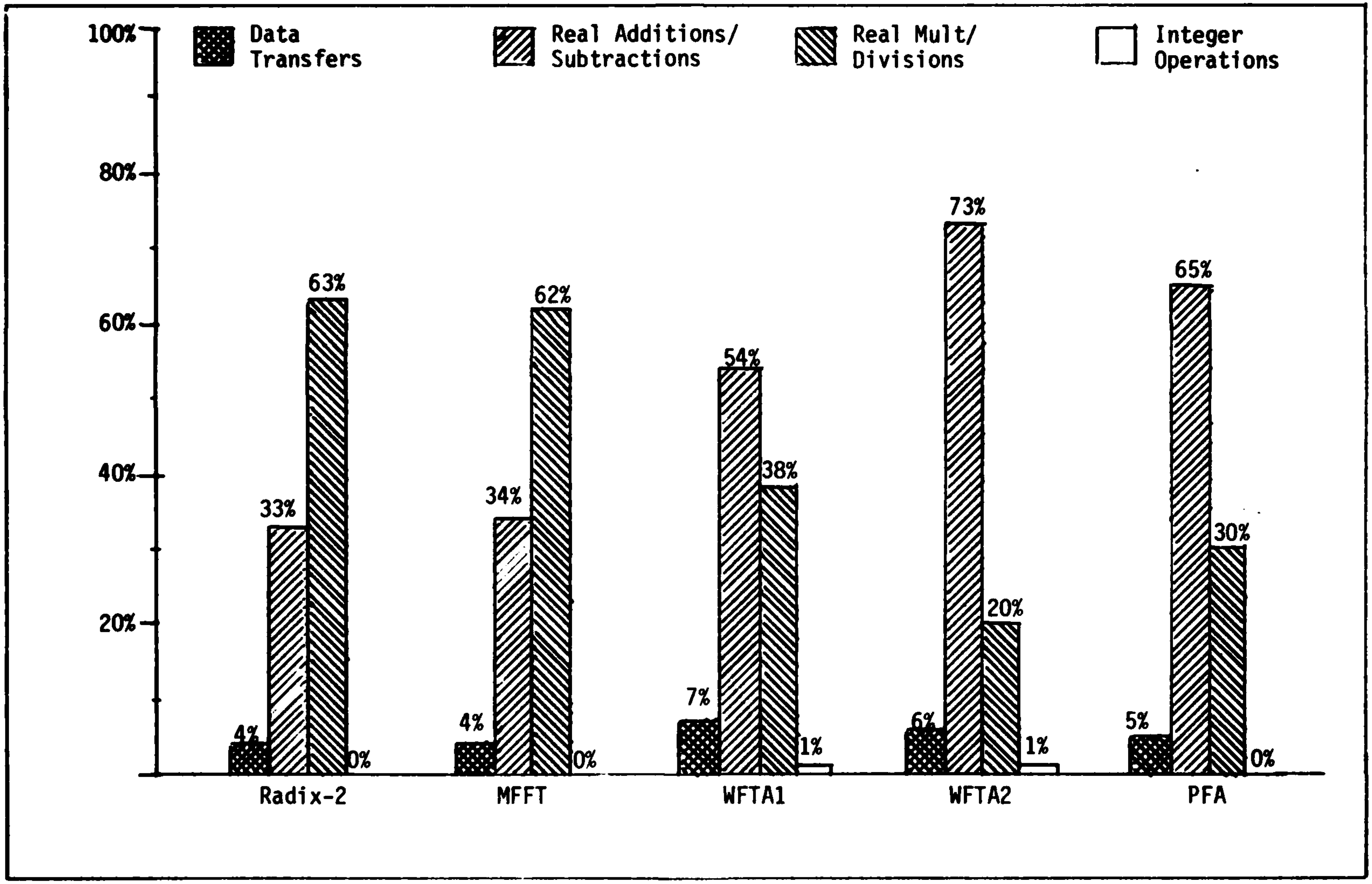


Figure 5.6. Percentage of Time per Instruction Category - Cromemco Z-2D

insufficient memory, but based solely on the number of floating multiplications, the time for WFTA1 is expected to be between the PFA and radix-2, while the time for WFTA2 is expected to be less than the time for the PFA. Since other microprocessors have the same basic architecture as the Cromemco Z-2D, the WFTA2 is expected to be the fastest algorithm and MFFT is expected to be the slowest algorithm on any microprocessor. However, the memory required for the WFTA is at least twice as large as the PFA. Thus, if the microprocessor is memory limited such that WFTA would not fit, the PFA would be the fastest algorithm.

VI. Comparison of Computer Architectures and Algorithms

As expected, the Cray-1 is the fastest computer, while the Cromemco Z-2D is the slowest. However, no one algorithm was always the fastest or slowest. Comparing the sequence lengths of 512, 1024, and 2048 with 504, 1008, and 2520, respectively, an ordering of the algorithms based on execution speed is different for each computer. The fastest algorithms on the different computers were: WFTA2 on the Cray-1, radix-2 on the Cyber 750, and PFA on the DEC VAX 11/780, IBM 370/155, DEC PDP 11/60, DEC PDP 11/50, and Cromemco Z-2D. The slowest algorithms were: WFTA1 on the Cray-1 and DEC VAX 11/780; MFFT on the IBM 370/155 and Cromemco Z-2D; and, depending on the sequence length, either WFTA1 or MFFT on the PDP 11/60 and PDP 11/50. Executing the algorithms on a faster computer did not increase the execution speeds of all the algorithms equally. For example, the radix-2 algorithm ran an average 2.55 times faster on the Cray-1 than on the Cyber 750, while the WFTA2 ran an average of 5.04 times faster. Table 6.1 lists the speed increases of one computer over another. Figure 6.1 shows a plot of the execution speed versus data transfers for the architectures studied. Table 6.2 lists the equations of the lines plotted in Figure 6.1. The reasons for the unequal increases in performance are related to the computer architecture and will be explored.

Comparing the Cyber 750 and the Cray-1, the WFTA1 had the greatest increase in execution speed. WFTA1 was an

TABLE 6.1

Average Speed Increase of

Algorithms for One Computer Over Another

Comparison	Radix-2	MFFT	WFTA1	WFTA2	PFA
Cray-1 over Cyber 750	2.55	4.90	2.71	5.04	3.65
Cyber 750 over VAX 11/780	16.29	6.81	10.16	9.61	7.12
VAX 11/780 over IBM 370/155	1.12	1.26	0.89	0.98	1.08
IBM 370/155 over PDP 11/60	1.36	1.28	1.82	1.84	1.72
PDP 11/60 over PDP 11/50	2.57	2.66	2.21	2.24	2.36
PDP 11/50 over Cromemco Z-2D	28.37	27.53	*	*	18.43

* Unable to execute due to insufficient memory.

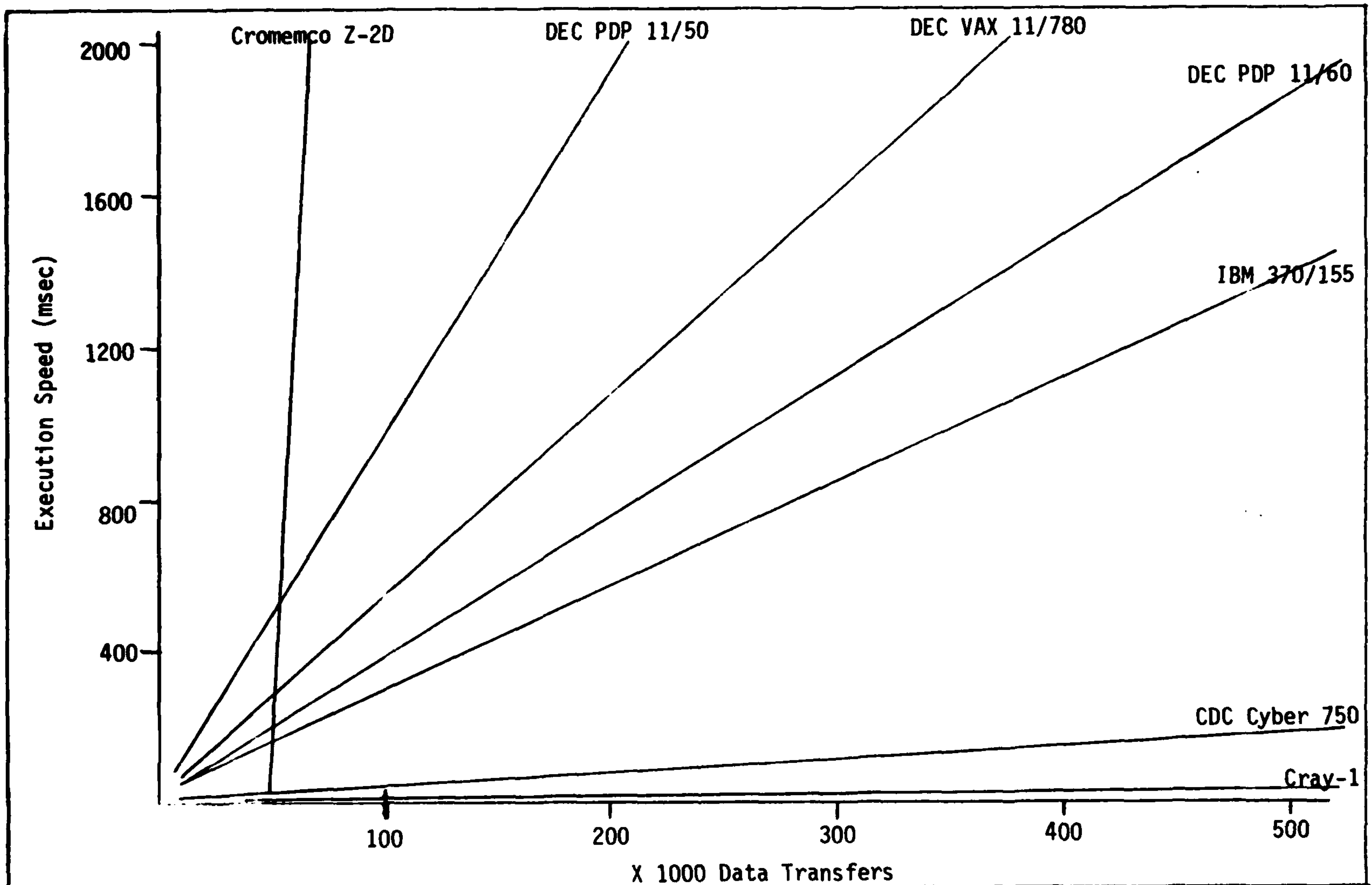


Figure 6.1. Execution Speed Versus Data Transfers

TABLE 6.2

Equations of Lines of Best Fit	
Cray-1:	$T = 0.0335\ D + 2.8403$
CDC Cyber 750:	$T = 0.3736\ D - 0.5944$
IBM 370/155:	$T = 2.6763\ D + 41.2501$
DEC VAX 11/780:	$T = 5.1865\ D + 38.7573$
DEC PDP 11/60:	$T = 3.6834\ D + 31.4450$
DEC PDP 11/50:	$T = 9.8966\ D - 2.0489$
Cromemco Z-2D:	$T = 97.3118\ D - 4576.0512$

average of 5.0 times faster on the Cray-1, while the radix-2 was only 2.6 times faster. The availability of vector operations benefited WFTA1, with its matrix structure, more than the others. Improvements in the Cyber 750 architecture could be made by decreasing the data transfer time through the use of high speed buffer storage. Optimization of algorithms for the Cyber 750 could be accomplished by minimizing the number of data transfers from memory.

In comparing the Cyber 750 and IBM 370/155, the speed increases of the Cyber over the IBM are approximately equal except for the radix-2 algorithm, which benefited twice as much as the others. Since the Cyber data transfer time is slower than its floating operation time, whereas the IBM data transfer time is faster than its floating operation time, algorithms with fewer data transfers see more of an improvement when executed on the Cyber. Thus, any improvement of the IBM 370/155 architecture for FFT execution should be directed towards increasing the throughput of floating operations. FFT algorithms can be optimized by decreasing the number of floating operations, even at the expense of increased data transfers.

In comparing the DEC VAX 11/780 to the IBM 370/155, the WFTA was fastest on the IBM, while the other three were faster on the VAX. The WFTA1 was an average of 1.1 times faster on the IBM, while the WFTA2 times were within 5% of each other for the two computers. The PFA is 1.08 times faster, MFFT is 1.26 times faster, and radix-2 is 1.12 times

faster on the VAX than the IBM. The instruction timings are not available for the VAX. However, the VAX has the greatest improvement over the IBM on the MFFT. Since the MFFT has a greater number of floating multiplies than the other algorithms, these results could be explained by the VAX having a smaller floating multiply to data transfer ratio than the IBM, which has a ratio of 7.4. Thus the VAX performs better than the IBM on algorithms with more floating point operations. However, performance is about equal on algorithms with a large number of data transfers. This is due to the similar architectures of the computers. Both have cache memories which improve the performance of algorithms with many data transfers.

The speed increases of the IBM 370/155 over the PDP 11/60 are the greatest in the WFTA and PFA. The WFTA2 is 1.8 times faster and the PFA is 1.7 times faster, while the MFFT is only 1.3 times faster. The ratio of the floating multiply to data transfer time is about 7.4 on the IBM 370/155, while on the PDP 11/60 it is only 2.0. Thus the IBM 370/155 performs better than the PDP 11/60 on algorithms which have fewer floating operations but more data transfers. No obvious areas of improvement can be found for the PDP 11/60 architecture. Likewise, optimization of FFT algorithms for the PDP 11/60 is not as simple as the other architectures. Data transfers can be traded for floating multiplies if eliminating on multiply adds less than two data transfers, since the speed ratio between these two instructions is approximately 2. The same reasoning can be

applied to trade-offs between data transfers/floating adds and floating adds/floating multiplies.

In comparing the PDP 11/60 and PDP 11/50, WFTA executes 2.2 times faster, while MFFT is 2.7 times faster. The ratio of floating multiply speed to data transfer speed is 2.7 for the PDP 11/50. Since the speed increases are relatively close, no relationship between the speed increases and the differences in the computer architectures can be made. The trade-offs discussed under the PDP 11/60 apply here with appropriate modifications to the instruction speed ratios.

The radix-2 algorithm shows the greatest increase in execution speed when run on the PDP 11/50 over the Cromemco Z-2D. It increased 28.37 times, while the PFA increased 18.43 times. The floating multiply to data transfer speed ratio is 308. Thus a greater improvement occurs in the algorithms with more floating operations. Since the correlation coefficients between the execution speed and the floating operations is the greatest, the Cromemco Z-2D architecture could best be improved with the addition of floating multiplication and addition functional units. Optimization of the algorithm could be accomplished by eliminating a floating multiplication as long as less than 308 data transfers are added.

VII. Minimum Computer Architecture for Efficient Performance

This section will present the minimum computer architecture needed to efficiently execute FFT algorithms. The proposed architecture is by no means optimum, because more features could be added to improve the algorithm performance if desired. The decision to include a feature into this architecture is based on the results from Chapters 5 and 6 of executing the given FFT algorithms on the seven computers. When implementing the architecture, at least the following three hardware features must be considered: functional units, local storage, and high speed buffer memory. But since architecture as defined here includes the compiler and operating system, the system software must also be considered. In addition, the optimization needed by a given architecture to improve its performance can be determined. The minimum hardware for an efficient FFT processor is shown in Figure 7.1.

Functional Units.

A functional unit is a hardware building block designed to perform a specific operation. Separate pipelined functional units, as in the CDC Cyber 750, decrease the dependence of the execution speed on the number of floating operations by allowing certain instructions to execute in parallel. The tables in Chapter 5 show that the greatest number of instructions are in the categories of integer addition, floating addition, and floating multiplication.

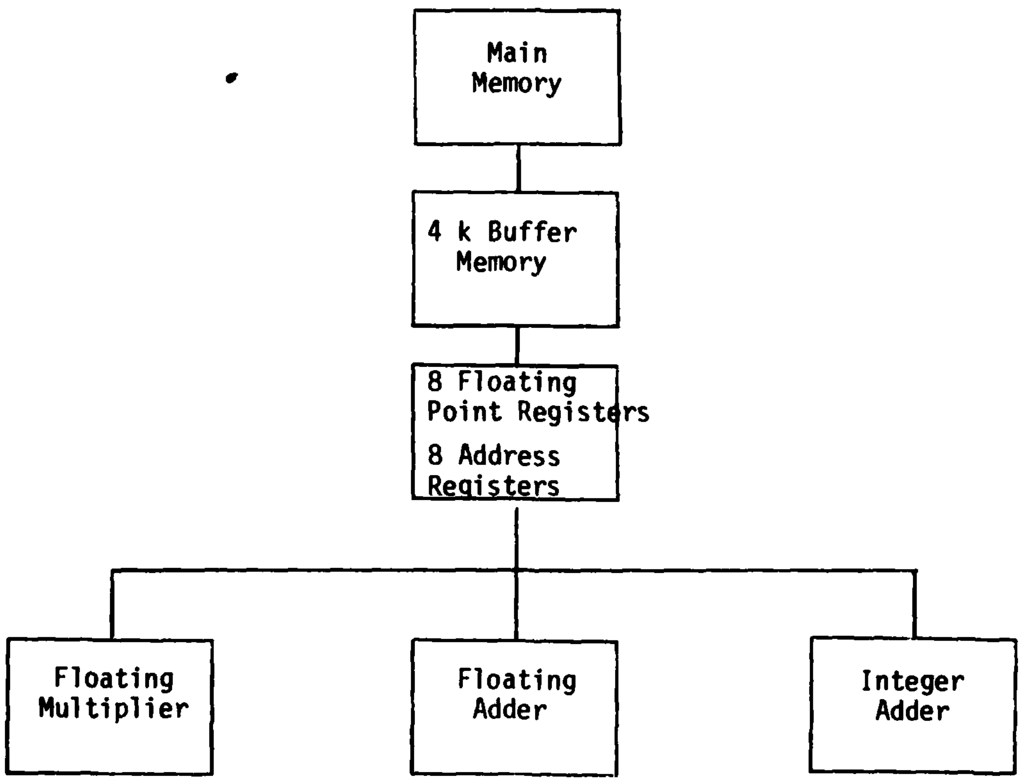


Figure 7.1. Minimum Hardware for Efficient Performance

Therefore, these are the minimum functional units required. Floating division always accounted for less than 0.1% of the floating instructions and therefore does not warrant a separate functional unit. Each of the functional units should be pipelined and have access to all of the high speed registers.

Local Storage.

Local storage consists of the high speed registers contained in the CPU. Local storage is used by the functional units to store operands and results, and also to hold the addresses of the operands. The floating point registers and the address registers need not be the same length. The length of the floating point registers corresponds to the accuracy of the digital representation, while the length of the address registers corresponds to the amount of memory used. Current architectures suggest that a minimum of 32 bits be used for the floating point registers and 16 bits for the address registers. However, longer length sequences may require more memory than 16 bits can address, thus the address registers usually should be longer. Increasing the number of high speed operand registers available to the functional units would decrease the number of data transfers required, and thus decrease the execution speed of the algorithm. The paper by Nawab and McClellan suggests that the number of registers necessary for the efficient implementation of FFT algorithms should be at least eight (Nawab and McClellan, 1979).

High Speed Buffer Memory.

The high speed buffer memory is between the local storage and main memory, both in terms of size and physical connections. All data passing between main memory and the registers must pass through the high speed buffer, or cache, memory. The data is retained in the cache memory until it is replaced by other data. The cache memory takes advantage of the fact that a program has a higher than average probability of referring to a piece of data or an instruction that it has recently referenced. The use of high speed buffer memory does not affect the number of data transfers, but it does reduce the average time to perform a data transfer. Ideally, the high speed buffer should be large enough to contain a typical algorithm loop, but with some algorithms that is not feasible. In general, a buffer memory of at least 4 kwords is necessary to obtain a 90% hit ratio (Baer, 1980).

System Software

In addition to the hardware employed, the compiler used by each machine must be considered. Compiler optimization will not affect the number of floating point operations, but it will affect the number of data transfers and integer operations. In an architecture such as this, which has several functional units, the compiler may reorder some of the operations to minimize the idle time of the functional units. The compiler must be matched to and take optimum advantage of the hardware. Improvements made on the

compiler could affect the number of data transfers and consequently the results presented here. The Cray-1 and VAX 11/780 used compilers that implemented the FORTRAN 77 language, while the others used compilers that implemented the older FORTRAN IV language. The newer language version has some changes that affect the performance of FFT algorithms. Specifically, FORTRAN 77 requires that all DO loops be checked for an initial value greater than the final value, and if that condition is found, the loop should not be executed at all. In the older FORTRAN IV, a loop was executed once regardless of the initial and final values. Thus the FORTRAN 77 compiler inserted extra code into the program that was not needed since the FFT programs had no such loops. These extra statements could only slow down the execution speed of the algorithms. Thus, for FFT algorithms, a FORTRAN IV compiler will produce better code than an equivalent FORTRAN 77 compiler. However, some FORTRAN 77 compilers, such as the one on the VAX 11/780, have an option to conform to the older standard. This option should be used wherever possible.

<u>Optimization of Current Architectures</u>

For a given currently available architecture, the areas needing improvement to optimize the architecture for FFT execution can be determined through the correlation coefficients. After each of the FFT algorithms has been run, with the execution speed measured and the instruction counts determined, the correlation coefficients between the execution speed and each major instruction category can be

calculated. Then the instruction category which has the largest correlation coefficient can be determined. The best improvement to the architecture would be one that reduces the execution time of that instruction category. For example, if the data transfers had the largest correlation coefficient, then features which improve the data transfer rate, such as high speed buffer memory, should be added. Alternatively, this information could be obtained by counting the instructions, as before, and then using the individual instruction times to obtain a calculated execution time. Then the correlation coefficients or the percentage of time spent on each instruction category can be determined. The instruction taking the largest percentage of time is the one that needs improvement. However, this method is not as good as using the correlation coefficients because it neglects the instruction overlap. Architectures in which a high correlation coefficient was measured between the execution speed and floating operations, as was done in a previous section for the Cromemco Z-2D, would benefit most from the addition of functional units. If the correlation coefficient is the greatest between a particular floating operation and the execution speed, then that operation has the greatest effect on the execution speed, and improvements should be made to increase the speed of that operation, which is normally done through the addition or improvement of functional units. Architectures with relatively slow data transfer times, which is indicated by a high

correlation coefficient between the execution speed and the data transfers, would benefit most from an increase in registers. The architectures that would benefit most from the addition of a high speed buffer memory are the ones with a high correlation coefficient between the execution speed and the data transfers, as in the CDC Cyber 750.

VIII. Prediction of Algorithm Performance

Knowledge of the computer architecture on which an algorithm will run can be used to predict which algorithm will have the minimum execution speed. To accomplish this, computer architectures can be divided into three different types: floating operation processors, data transfer processors, and vector processors.

Floating Operation Processors.

Floating operation processors are those which execute floating operations well and whose execution speed is limited mainly by the data transfer rates of the operands. These architectures typically have a data transfer time which is greater than half of the floating operation time and a high correlation coefficient between the execution speed and number of data transfers. The CDC Cyber 750 is an example of a floating operation processor. These architectures execute algorithms with a minimum number of data transfers most efficiently. Therefore, ranking the algorithms according to the number of data transfers, the radix-2 algorithm would have the minimum execution speed, followed by the PFA, WFTA, and MFFT. In fact, the execution speed could be roughly predicted using only knowledge of the number of data transfers, independent of the algorithm used. For example, the execution speed for the algorithms run on the CYBER 750 are plotted against the number of data transfers in Figure 8.1. The execution speed is directly proportional to the number of data transfers. The equation

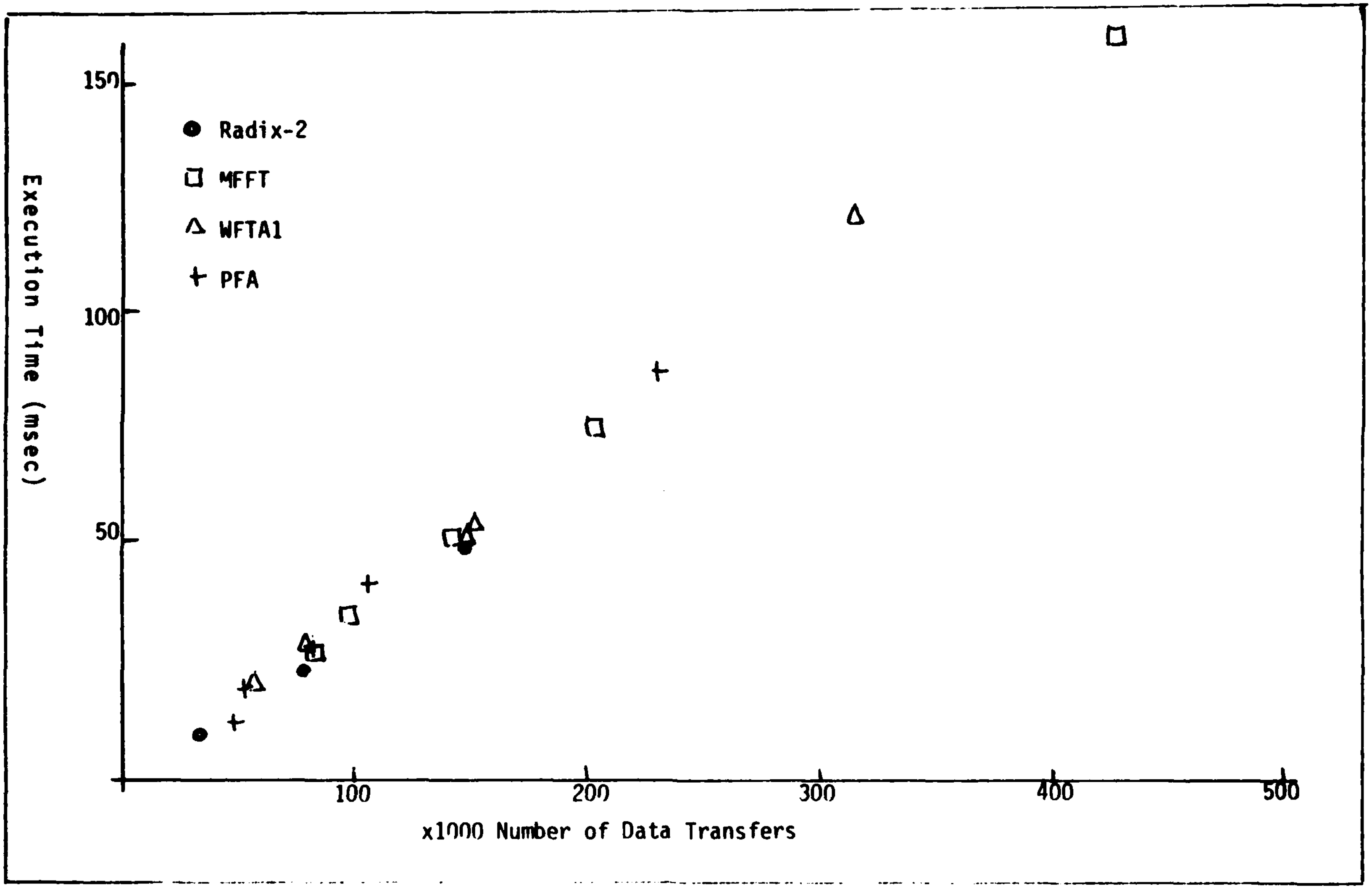


Figure 8.1. Cyber Execution Speed Versus Data Transfers

for the line of best fit is

$$T = 0.3736\ D - 0.5944 \qquad (8\text{-}1)$$

where T is the time in milliseconds and D is the number of data transfers in thousands. The average error between the points and this line of best fit is 4.1%. However, the proportionality constant will vary with different computers, but is closely related to the data transfer rate of the computer. Thus, for an architecture like the Cyber 750, the fastest algorithm is the one with the fewest data transfers for that particular sequence length.

Data Transfer Processors.

Data transfer processors have their execution speed limited by the speed of their floating point operations. These processors have a single multipurpose functional unit and usually a high speed buffer. The IBM 370/155 and Cromemco Z-2D are examples of data transfer processors. The correlation coefficient between the execution speed and the floating operations is the greatest. Thus, the number of data transfers does not predict the execution speed as well as in the floating operations processors, as can be seen in Figure 8.2. The equation for the line of best fit is

$$T = 2.6736\ D + 41.2501 \qquad (8\text{-}2)$$

where T is the time in milliseconds and D is the number of data transfers in thousands. The average percentage error between the actual execution times and those predicted by the line of best fit is 16.0%, which is about 4 times greater than the error for the floating operations processors. Data transfer processors have a data transfer

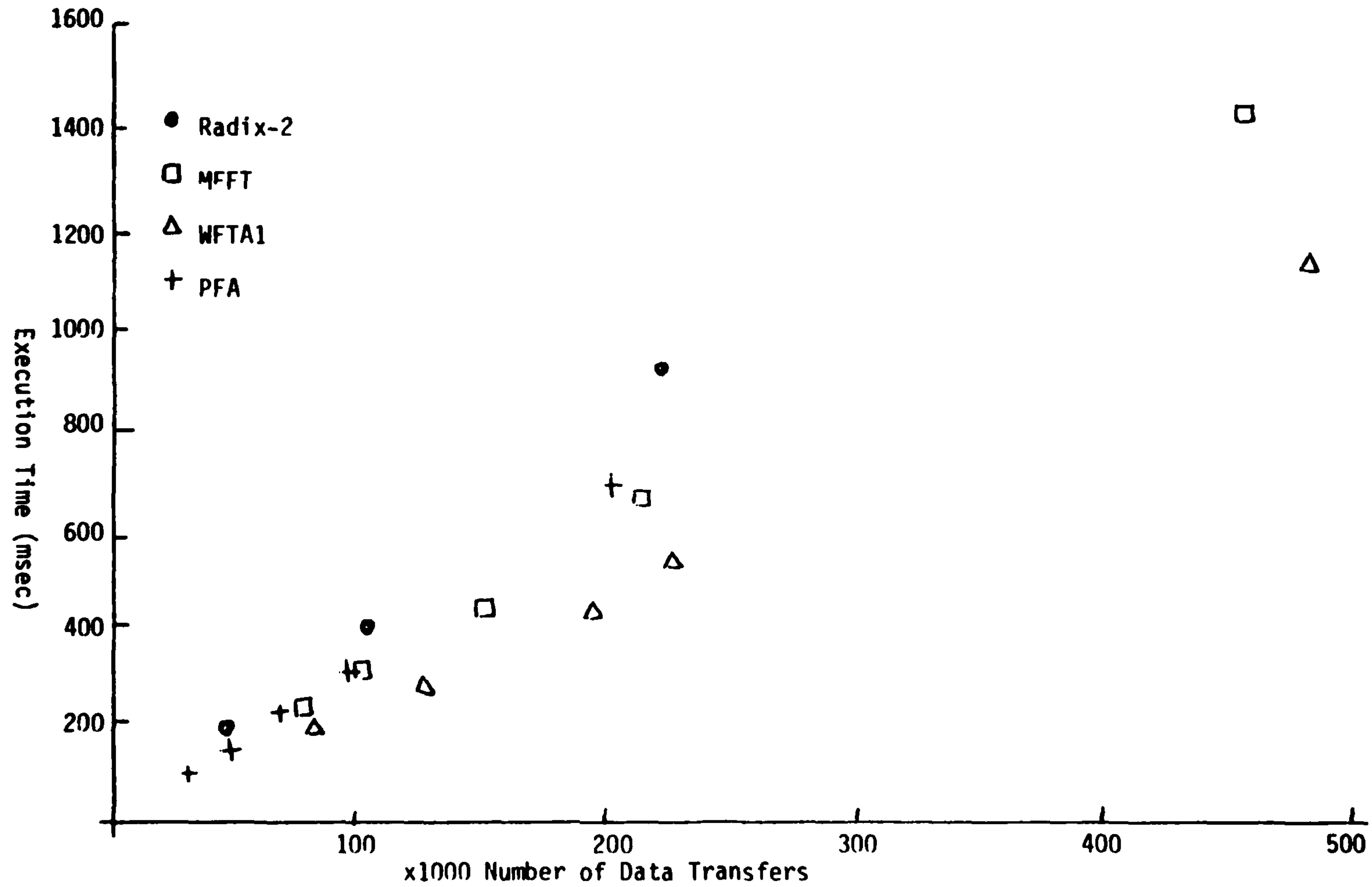


Figure 8.2. IBM Execution Speed Versus Data Transfers

time which is less than half of the floating operation time, and execute algorithms with fewest floating operations most efficiently. For this architecture, the PFA would be the fastest architecture, followed by the radix-2, MFFT, and WFTA, with the particular order of the last three depending on the ratio of floating add speed to floating multiply speed.

Vector Processors.

Vector processors, or array processors, have functional units specifically for vector operations. The Cray-1 is an example of a vector processor. Vector processors execute the initialized WFTA, with its nested structure, most efficiently, followed by the PFA, radix-2, and MFFT, with the order of the last three dependent on the other features available in the processor.

Now that guidelines for predicting algorithm performance, based on the knowledge of the computer architecture, have been proposed, the next chapter will summarize the results and conclusions of this study.

IX. Conclusions and Recommendations

Conclusions

This study presented an evaluation of the four major FFT algorithms on seven different computers. The data clearly shows that no one algorithm is faster than another in all cases. The reasons why certain algorithms perform better on certain computers were explained in terms of the computer architecture. The execution speeds were related to four different instruction categories: floating add/subtract, floating multiply/divide, integer operations, and data transfers. The average correlation coefficients were 0.9518, 0.8614, 0.9401, and 0.9792, respectively. In all cases, the number of data transfers was highly correlated with the execution speeds. For floating operation processors, the number of data transfers was a better predictor of the algorithm performance, with a 4.1% error, than the number of floating operations. In computer architectures with several pipelined functional units, the number of data transfers had a major impact on the execution speed, and the radix-2 was the fastest algorithm. In computer architectures with a single multipurpose functional unit, such as microcomputers, floating operations had a major impact on execution speed, but data transfers were still important. These architectures executed the PFA fastest. Architectures with vector operations executed the initialized WFTA fastest.

The minimum computer architecture for the time

efficient execution of FFT algorithms was presented. This architecture included several functional units and a high speed buffer memory. A qualitative method was given for predicting the performance of FFT algorithms based on the computer architecture. This information can be used to determine which algorithm should be used for a particular computer, and also to determine the computer architecture for dedicated FFT processors. The correlation coefficients between the different instructions and the execution speeds can be used to determine which areas of the computer architecture need improvement in order to optimize FFT algorithm execution.

Recommendations

To complete this study, the instruction timings of the VAX 11/780 need to be determined. These instruction timings can then be used to determine the percentage of time spent on each instruction category.

The effects of the FORTRAN compilers on the execution speeds need to be studied in more detail. The characteristics of a good FFT compiler, and operating system, need to be determined, as well as the optimizations that should be performed to improve FFT execution speed. The FORTRAN code should be compared to the same algortihm written in assembly language.

To properly study the effects of the computer architecture on FFT algorithm performance, simulation programs are needed for each architecture. However, no such programs were available at the time of this study. In

addition, a simulation program of the proposed minimum architecture should be developed. The results of the simulation program could be used to make improvements to the proposed minimum architecture. These simulation programs should account for instruction overlaps and cache memory hits.

A quantitative method for predicting performance needs to be developed. This model should be based on the architectural features and the algorithm and should result in a predicted execution time.

The analysis presented in this study needs to be expanded to include dedicated FFT processors and array processors. The results could be used to improve the design and thus the performance of dedicated FFT processors.

Bibliography

Baer, J. L. Computer Systems Architecture. Rockville, MD: Computer Sciences Press, 1980.

Blanken, J.D. and P. L. Rustan. "Selection Criteria for Efficient Implementation of FFT Algorithms," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-30, pp. 107-109, Feb 1982.

Burrus, C. S. and P. W. Eschenbacher. "An In-Place, In-Order Prime Factor FFT Algorithm," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-29, pp. 806-817, Aug. 1981.

Chu, Shuni, and C. Sidney Burrus. "A Prime Factor FFT Algorithm Using Distributed Arithmetic," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-30, pp. 217-226, Apr. 1982.

Control Data Corporation. CDC Cyber 170 Computer Systems Models 171 through 175 (Levels A, B, C) and Model 176 (Level A) Hardware Reference Manual, Manual 60420000, Control Data Corporation, 1979.

Control Data Corporation. FORTRAN Extended Version 4 Reference Manual, Revision H, Manual 60497800, Control Data Corporation, 1982.

Cooley, J. W. and J. W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, Vol. 19, pp. 297-301, Apr. 1965.

Cray Research, Inc. Cray-1 Computer System Hardware Reference Manual, Publication 2240004.

Digital Equipment Corporation. PDP-11 Processor Handbook, 1976.

Digital Equipment Corporation. PDP-11 Processor Handbook, 1979.

Digital Equipment Corporation. PDP-11 Software Handbook, 1981.

Digital Equipment Corporation. VAX Architecture Handbook, 1981.

Digital Equipment Corporation. VAX Hardware Handbook, 1982.

Dixon, Grant. Digital Equipment Corp. VAX Representative, Dayton, OH., Private Communication.

International Business Machines. IBM OS FORTRAN IV (H Extended) Compiler Programmer's Guide, SC28-6852-2, IBM, 1974.

International Business Machines. IBM System 370 Model 155 Functional Characteristics, GA22-6942-2, IBM, May 1972.

Johnson, H. W., and C. S. Burrus. "The Design of Optimal DFT Algorithms Using Dynamic Programming," Proceedings of the 1982 International Conference on Acoustics, Speech, and Signal Processing, pp. 20-23, New York: Institute of Electrical and Electronics Engineers, 1982.

Kernighan, Brian W. UNIX for Beginners, 2 ed., Murray Hill, New Jersey: Bell Labs, 1978.

Kolba, D. P. and T. W. Parks. "A Prime Factor FFT Algorithm Using High-Speed Convolution," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-25, pp. 281-294, Aug. 1977.

McClellan, J. H. and H. Nawab. "Complex General-N Winograd Fourier Transform Algorithm (WFTA)," Programs for Digital Signal Processing, IEEE Press, New York: John Wiley and Sons, Inc., 1979.

Morris, L. R. "A Comparative Study of Time Efficient FFT and WFTA Programs for General Purpose Computers," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-26, pp. 141-150, Apr. 1978.

Nawab, H. and J. H. McClellan. "Bounds on the Minimum Number of Data Transfers in WFTA and FFT Programs," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-27, pp. 394-398, Aug. 1979.

Oppenheim, A. V. and R. W. Schafer. Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1975.

Preuss, Robert D. "Very Fast Computations of the Radix-2 Discrete Fourier Transform," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-30, pp. 595-607, Aug. 1982.

Rabiner, L. R., and B. Gold. Theory and Application of Digital Signal Processing. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1975.

Route, G. P. "Efficient Computer Architectures for Computing Discrete Fourier Transforms," M. S. Thesis, Air Force Institute of Technology, 317 pp., Dec. 1981.

Russel, Richard M. "The Cray-1 Computer System," Communications of the ACM, 21: 66, Jan. 1978.

Silverman, H. F. "An Introduction to Programming the Winograd Fourier Transform Algorithm (WFTA)," IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-25, pp. 152-165, Apr. 1977.

Singleton, R. C. "Mixed Radix Fast Fourier Transforms," Programs for Digital Signal Processing, IEEE Press, New York: John Wiley and Sons, Inc., 1979.

Thrall, R. M. and L. Tornheim. Vector Spaces and Matrices. New York: Wiley, 1957.

Zilog, Inc. Z80 CPU Technical Manual. Cupertino, CA: Zilog, Inc., 1977.

APPENDIX A

Radix-2 Algorithm Program

This appendix contains an in-place FFT program used in executing and timing the radix-2 algorithm. The program was originally developed by Rabiner and Gold (Rabiner and Gold, 1975) and was modified by Route to eliminate the calls to the complex library function (Route, 1981). The program consists of a driver and a single subroutine. The subroutine is called using the format

CALL FFT2CM(A,B,M,XIN,XOUT)

where the variables are defined as follows.

A -A is a real array specifying the real components of the input sequence x(n) and the output sequence X(k).

B -B is a real array specifying the imaginary components of the input sequence x(n) and the output sequence X(k).

M -M is an integer specifying the power to which two is raised to obtain the sequence length.

XIN -XIN is a real value set to the starting value of the realtime clock.

XOUT -XOUT is a real value set to the difference between the final value of the realtime clock and XIN.

```
C
C**********************************************************************
C
C PROGRAM RADIX 2 FFT DRIVER
C
C  THIS IS A DRIVER PROGRAM FOR THE SINGLETON FAST FOURIER TRANSFORM
C  ROUTINE. IT WAS ORIGINALLY WRITTEN TO BE RUN ON THE PDP 11/50
C  OF AFWAL/FIGX.  IT IS WRITTEN IN FORTRAN AND IS TO BE COMPILED
C  WITH THE DEC F4P COMPILER
C
C  AUTHOR:  MARK A. MEHALIC
C  DATE:  21 APR 83
C  VERSION:  1.0
C  SUBROUTINES CALLED:  FFT2CM,GRAPH
C
C**********************************************************************
C
C
C**********************************************************************
C
C  DECLARE ALL VARIABLES AND ARRAYS USED
C
C**********************************************************************
C
      REAL A(512),B(512)
      REAL MAG(512)
C
C**********************************************************************
C
C  SET UP THE CONSTANTS USED IN THE PROGRAM
C
C**********************************************************************
C
      N=512
      M = 9
      PI=3.14159265359
      T1=0.01
      E = 2.71828
      T=0.0
C
C**********************************************************************
C
C  DIGITIZE THE CONTINUOUS FUNCTION
C
C**********************************************************************
C
      DO 10 I=1,N
         A(I)=(E**(-T)) * COS(50*PI*T)
         B(I)=0.0
         T=T+T1
10    CONTINUE
      CALL GRAPH(A,N)
C
```

```
C**********************************************************************
C
C  CALL THE SUBROUTINE FFT2CM (MODIFIED RADIX 2 ALGORITHM)
C
C**********************************************************************
C
      CALL FFT2CM(A,B,N,XIN,XOUT)
C
C**********************************************************************
C
C  PRINT OUT THE EXECUTION TIME
C
C**********************************************************************
C
      PRINT*, 'THE EXECUTION TIME IS',XOUT
C
C**********************************************************************
C
C  DETERMINE THE MAGNITUDE OF THE FOURIER TRANSFORM
C
C**********************************************************************
C
      DO 20 I=1,N
         MAG(I)=((A(I))**2+(B(I))**2)**0.5
         PRINT*, A(I), B(I), MAG(I)
20    CONTINUE
C
C**********************************************************************
C
C  CALL GRAPH WITH THE MAGNITUDE ARRAY AND NUMBER OF POINTS
C
C**********************************************************************
C
      CALL GRAPH(MAG,N)
C
C**********************************************************************
C
C  END OF MAIN PROGRAM
C
C**********************************************************************
C
      END
```

```
C
C*****************************************************************************
C
C  SUBROUTINE FFT RADIX-2 FOR PDP 11/50
C
C  AUTHOR:        GARY P. ROUTE
C  CODED BY:      MAR  A. MEHALIC
C  DATE:          21 APR 83
C  VERSION:       1.0
C
C  THIS SUBROUTINE CALCULATES THE FAST FOURIER TRANSFORM OF A SEQUENCE
C       WHOSE LENGTH IS A MULTIPLE OF 2.  IT HAS BEEN SPECIFICALLY
C       MODIFIED FOR THE PDP 11 SERIES OF COMPUTERS SO THAT THE CALL
C       TO THE COMPLEX FUNCTIONS OF THE LIBRARY HAVE BEEN ELIMINATED.
C       AS A RESULT, IT WILL RUN FASTER THAN THE FFT2C ALGORITHM.
C
C*****************************************************************************
C
      SUBROUTINE FFT2CM(A,B,M,XIN,XOUT)
C
C*****************************************************************************
C
C  DIMENSION THE INPUT (AND OUTPUT) ARRAYS FOR THE SEQUENCE LENGTH
C
C*****************************************************************************
C
      DIMENSION A(512), B(512)
      XIN=SECNDS(0.0)
      N=2**M
      NV2=N/2
      NM1=N-1
      J=1
      DO 7 I=1,NM1
      IF (I.GE.J) GO TO 5
      TR = A(J)
      TI = B(J)
      A(J) = A(I)
      B(J) = B(I)
      A(I) = TR
      B(I) = TI
5     K = NV2
6     IF (K.GE.J) GO TO 7
      J = J-K
      K = K/2
      GO TO 6
7     J = J+K
      PI = 3.1415926535898
      DO 20 L=1,M
         LE = 2**L
         LE1 = LE/2
         UR = 1.0
         UI = 0.0
         WR = COS(PI/LE1)
```

```
      UI = SIN(PI/LE1)
      DO 20 J=1,LE1
            DO 10 I=J,N,LE
                  IP = I+LE1
                  TR = A(IP)*UR-B(IP)*UI
                  TI = A(IP)*UI+B(IP)*UR
                  A(IP) = A(I)-TR
                  B(IP) = B(I)-TI
                  A(I) = A(I)+TR
10                B(I) = B(I)+TR
            TR = UR*UR-UI*UI
            UI = UR*UI+UI*UR
20          UR = TR
C
C***********************************************************************
C
C  CALCULATE THE EXECUTION TIME
C
C***********************************************************************
C
      XOUT = SECNDS(XIN)
C
C***********************************************************************
C
C  END OF SUBROUTINE
C
C***********************************************************************
C
      RETURN
      END
```

APPENDIX B

Singleton's Algorithm Program

This appendix contains the driver and subroutine programs used to execute and time the Singleton mixed radix algorithm. The algorithm is divided into two subroutines: FFTSNG and FFTMX. FFTSNG does the sequence length factorization and calls FFTMX, which does the short transforms. The subroutines are called using the format

```
CALL FFTSNG(A,B,NSEG,N,NSPN,ISN,AT,CK,BT,SK,KD,NP,NPM,
                                                XIN,XOUT)
```

and

```
CALL FFTMX(A,B,NTOT,NF,NSPAN,ISN,M,KT,AT,CK,BT,SK,KD,NP,
                                                NPM,NFAC)
```

where the variables are defined as follows.

N -N is the integer sequence length.

A -A is a real array of dimension N. When FFTSNG is called, A contains the real components of the input sequence x(n). On return from FFTNSG, A contains the real components of the output sequence X(k).

B -B is a real array of dimension N. When FFTSNG is called, B contains the imaginary components of the sequence x(n). On return from FFTSNG, B contains the imaginary part of the output sequence X(k).

NSEG -NSEG is an integer equal to the total number of complex data values divided by (N*NSPN). Usually, NSEG=1.

NSPN -NSPN is an integer specifying the spacing of

consecutive data values. Usually, NSPN=1.

ISN -ISN is an integer specifying the direction of the FFT. If ISN is negative, a forward transform is performed. If ISN is positive, an inverse transform is performed scaled by 1/N.

AT,BT -AT and BT are real arrays dimensioned as the maximum of 1.) the largest odd prime factor, or 2.) the product of the square factors of N. They are used for temporary storage during the transform of an odd prime factor greater than 5 and during the permutation of the square free factors.

CK,SK -CK and SK are real arrays dimensioned at least equal to the largest odd prime factor. They are used to provide temporary storage during the transform of an odd prime factor greater than 5.

KD -KD is an integer specifying the number of square factors in N.

NP -NP is an integer array of dimension at least NPM. It is used to store the digit reversed order of the square free factors.

NPM -NPM is an integer value equal to the product of the square free factors of N.

XIN -XIN is a real value used to hold the initial value of the real time clock.

XOUT -XOUT is a real value containing the amount of time required for the subroutine to execute.

```
      DIMENSION A(2520),B(2520)
      DIMENSION XAVG(4)
      DIMENSION AT(7),CK(7),BT(7),SK(7),NP(70)
      NSEG = 1
      NSPN = 1
      ISN = -1
      KD = 7
      NPM = 70
      READ(5,*) NUMTIM
      DO 999 INUM=1,NUMTIM
      READ(5,*) T,DELT,LIST
C     WRITE(6,601) T,DELT,LIST
601   FORMAT(1H1,'TEST WITH T =',F10.2,' DELT = ',F10.2,'LIST OPT=',I3)
      LCOUNT = 0
111   CONTINUE
      LCOUNT = LCOUNT + 1
      NPTS = (T/DELT) + 0.4
      DELF = 1.0/T
      PI = 3.14159265358
      E = 2.71828
      F0 = 25.0
      DO 10 I=1,NPTS
        T1 = (I-1)*DELT
        ARG = 2.0*PI*F0*T1
        TE = -1.0*T1
        B(I) = 0.0
10      A(I) = (E**TE)*COS(ARG)
      CALL FFTSNG(A,B,NSEG,NPTS,NSPN,ISN,AT,CK,BT,SK,KD,NP,NPM,XIN,XOUT)
      PRINT*,'XOUT',XOUT
      XAVG(LCOUNT) = XOUT
      DO 30 I=1,NPTS
30      A(I) = ((A(I)**2) + (B(I)**2)) ** 0.5
      PRINT*, NPTS
      IF(LCOUNT.LT.3) GO TO 111
      TAVG = 0.
      DO 599 J=1,3
        TAVG = TAVG + XAVG(J)
599     CONTINUE
      TAVG = TAVG/3
      PRINT*, 'AVG XOUT = ',TAVG
      IF(LIST.NE.1) GO TO 999
      WRITE(6,600)
600   FORMAT('1',20X,'MAG',16X,'IMAG',16X,16X,/)
      WRITE(6,700) (I,A(I),B(I),I=1,NPTS)
700   FORMAT(5X,I5,5X,E14.7,5X,E14.7,5X)
999   CONTINUE
      STOP
      END
```

```
C
C*****************************************************************************
C
C  SUBROUTINE SINGLETON MIXED RADIX FFT ALGORITHM
C
C  THIS SUBROUTINE IS TAKEN FROM 'PROGRAMS FOR DIGITAL SIGNAL
C       PROCESSING' PUBLISHED BY THE IEEE.
C
C*****************************************************************************
C
C
C*****************************************************************************
C
C  THE SUBROUTINE STATEMENT WAS MODIFIED TO INCLUDE THE INPUT AND
C       OUTPUT TIMES ON 21 APR 83.  IN ADDITION, THE CALL TO THE
C       SECONDS SUBROUTINE TO CALCULATE EXECUTION TIME WAS ADDED.
C
C*****************************************************************************
C
      SUBROUTINE FFTSNG(A,B,NSEG,N,NSPN,ISN,AT,CK,BT,SK,KD,NP,NPM,
     1XIN,XOUT)
C
C
C                              THE COMMENTS PRECEDED BY  CC  ARE THE
C                              ORIGINAL CARDS . THE CHANGES WERE MADE
C                              BECAUSE THIS FORTRAN COMPILER CANNOT
C                              ALLOCATE AND DE-ALLOCATE MEMORY
C
C
C
C-----------------------------------------------------------------------
C SUBROUTINE:  FFT
C MULTIVARIATE COMPLEX FOURIER TRANSFORM, COMPUTED IN PLACE
C USING MIXED-RADIX FAST FOURIER TRANSFORM ALGORITHM.
C-----------------------------------------------------------------------
C
C
C ARRAYS A AND B ORIGINALLY HOLD THE REAL AND IMAGINARY
C      COMPONENTS OF THE DATA, AND RETURN THE REAL AND
C      IMAGINARY COMPONENTS OF THE RESULTING FOURIER COEFFICIENTS.
C MULTIVARIATE DATA IS INDEXED ACCORDING TO THE FORTRAN
C      ARRAY ELEMENT SUCCESSOR FUNCTION, WITHOUT LIMIT
C      ON THE NUMBER OF IMPLIED MULTIPLE SUBSCRIPTS.
C      THE SUBROUTINE IS CALLED ONCE FOR EACH VARIATE.
C      THE CALLS FOR A MULTIVARIATE TRANSFORM MAY BE IN ANY ORDER.
C
C N IS THE DIMENSION OF THE CURRENT VARIABLE.
C NSPN IS THE SPACING OF CONSECUTIVE DATA VALUES
C      WHILE INDEXING THE CURRENT VARIABLE.
C NSEG*N*NSPN IS THE TOTAL NUMBER OF COMPLEX DATA VALUES.
C THE SIGN OF ISN DETERMINES THE SIGN OF THE COMPLEX
C      EXPONENTIAL, AND THE MAGNITUDE OF ISN IS NORMALLY ONE.
C      THE MAGNITUDE OF ISN DETERMINES THE INDEXING INCREMENT FOR A&B.
```

```
C
C IF FFT IS CALLED TWICE, WITH OPPOSITE SIGNS ON ISN, AN
C      IDENTITY TRANSFORMATION IS DONE...CALLS CAN BE IN EITHER ORDER.
C      THE RESULTS ARE SCALED BY 1/N WHEN THE SIGN OF ISN IS POSITIVE.
C
C A TRI-VARIATE TRANSFORM WITH A(N1,N2,N3), B(N1,N2,N3)
C IS COMPUTED BY
C        CALL FFT(A,B,N2*N3,N1,1,-1)
C        CALL FFT(A,B,N3,N2,N1,-1)
C        CALL FFT(A,B,1,N3,N1*N2,-1)
C
C A SINGLE-VARIATE TRANSFORM OF N COMPLEX DATA VALUES IS COMPUTED BY
C        CALL FFT(A,B,1,N,1,-1)
C
C THE DATA MAY ALTERNATIVELY BE STORED IN A SINGLE COMPLEX
C      ARRAY A, THEN THE MAGNITUDE OF ISN CHANGED TO TWO TO
C      GIVE THE CORRECT INDEXING INCREMENT AND A(2) USED TO
C      PASS THE INITIAL ADDRESS FOR THE SEQUENCE OF IMAGINARY
C      VALUES, E.G.
C        CALL FFT(A,A(2),NSEG,N,NSPN,-2)
C
C ARRAY NFAC IS WORKING STORAGE FOR FACTORING N.  THE SMALLEST
C      NUMBER EXCEEDING THE 15 LOCATIONS PROVIDED IS 12,754,584.
C
      DIMENSION A(N),B(N),NFAC(15),AT(KD),BT(KD),CK(KD),SK(KD),NP(NPM)
C
CC    COMMON /CSTAK/ DSTAK(2500)
CC    DOUBLE PRECISION DSTAK
CC    INTEGER ISTAK(5000)
CC    REAL RSTAK(5000)
C
CC    EQUIVALENCE (DSTAK(1),ISTAK(1))
CC    EQUIVALENCE (DSTAK(1),RSTAK(1))
C
C DETERMINE THE FACTORS OF N
C
      XIN=SECNDS(0.0)
      M = 0
      NF = IABS(N)
      K = NF
      IF (NF.EQ.1) RETURN
      NSPAN = IABS(NF*NSPN)
      NTOT = IABS(NSPAN*NSEG)
      IF (ISN*NTOT.NE.0) GO TO 20
C     IERR = I1MACH(4)
C     WRITE (IERR,9999) NSEG, N, NSPN, ISN
C999  FORMAT (31H ERROR - ZERO IN FFT PARAMETERS, 4I10)
      PRINT*,'ISN*TOT.NE.0'
      RETURN
C
   10 M = M + 1
      NFAC(M) = 4
      K = K/16
   20 IF (K-(K/16)*16.EQ.0) GO TO 10
```

```
      J = 3
      JJ = 9
      GO TO 40
   30 M = M + 1
      NFAC(M) = J
      K = K/JJ
   40 IF (MOD(K,JJ).EQ.0) GO TO 30
      J = J + 2
      JJ = J**2
      IF (JJ.LE.K) GO TO 40
      IF (K.GT.4) GO TO 50
      KT = M
      NFAC(M+1) = K
      IF (K.NE.1) M = M + 1
      GO TO 90
   50 IF (K-(K/4)*4.NE.0) GO TO 60
      M = M + 1
      NFAC(M) = 2
      K = K/4
C ALL SQUARE FACTORS OUT NOW, BUT K .GE. 5 STILL
   60 KT = M
      MAXP = MAX0(KT+KT+2,K-1)
      J = 2
   70 IF (MOD(K,J).NE.0) GO TO 80
      M = M + 1
      NFAC(M) = J
      K = K/J
   80 J = ((J+1)/2)*2 + 1
      IF (J.LE.K) GO TO 70
   90 IF (M.LE.KT+1) MAXP = M + KT + 1
      IF (M+KT.GT.15) GO TO 120
      IF (KT.EQ.0) GO TO 110
      J = KT
  100 M = M + 1
      NFAC(M) = NFAC(J)
      J = J - 1
      IF (J.NE.0) GO TO 100
C
  110 MAXF = M - KT
      MAXF = NFAC(MAXF)
      IF (KT.GT.0) MAXF = MAX0(NFAC(KT),MAXF)
CC    J = ISTKGT(MAXF*4,3)
CC    JJ = J + MAXF
CC    J2 = JJ + MAXF
CC    J3 = J2 + MAXF
CC    K = ISTKGT(MAXP,2)
CC    K = ISTKGT(MAXP,2)
CC    CALL FFTMX(A, B, NTOT, NF, NSPAN, ISN, M, KT, RSTAK(J),
CC   *    RSTAK(JJ), RSTAK(J2), RSTAK(J3), ISTAK(K), NFAC)
      CALL FFTMX(A,B,NTOT,NF,NSPAN,ISN,M,KT,AT,CK,BT,SK,KD,NP,NPM,NFAC)
CC    CALL ISTKRL(2)
C
C*****************************************************************************
C
```

```
C  CODE ADDED TO CALCULATE EXECUTION TIME -- 21 APR 83
C
C*******************************************************************************
C
      XOUT=SECNDS(XIN)
      RETURN
C
C120  IERR = I1MACH(4)
C     WRITE (IERR,9998) N
C998  FORMAT (50H ERROR - FFT PARAMETER N HAS MORE THAN 15 FACTORS-,
120   PRINT*,'N HAS MORE THAN 15 FACTORS'
      RETURN
      END
C
      SUBROUTINE FFTMX(A,B,NTOT,N,NSPAN ,ISN,M,KT,AT,CK,BT,SK,
     CKD,NP,NPM,NFAC)
C-----------------------------------------------------------------------
C SUBROUTINE:  FFTMX
C CALLED BY SUBROUTINE 'FFT' TO COMPUTE MIXED-RADIX FOURIER TRANSFORM
C-----------------------------------------------------------------------
C
CC    SUBROUTINE FFTMX(A,B,NTOT,NF,NSPAN,ISN,M,KT,NFAC)
C
      DIMENSION A(N),B(N),AT(KD),BT(KD),CK(KD),SK(KD),NP(NPM)
      DIMENSION NFAC(15)
C
      INC = IABS(ISN)
      NT = INC*NTOT
      KS = INC*NSPAN
      RAD = ATAN(1.0)
      S72 = RAD/0.625
      C72 = COS(S72)
      S72 = SIN(S72)
      S120 = SQRT(0.75)
      IF (ISN.GT.0) GO TO 10
      S72 = -S72
      S120 = -S120
      RAD = -RAD
      GO TO 30
C
C SCALE BY 1/N FOR ISN .GT. 0
C
   10 AK = 1.0/FLOAT(N)
      DO 20 J=1,NT,INC
        A(J) = A(J)*AK
        B(J) = B(J)*AK
   20 CONTINUE
C
   30 KSPAN = KS
      NN = NT - INC
      JC = KS/N
C
C SIN, COS VALUES ARE RE-INITIALIZED EACH LIM STEPS
C
```

```
      LIM = 32
      KLIM = LIM*JC
      I = 0
      JF = 0
      MAXF = M - KT
      MAXF = NFAC(MAXF)
      IF (KT.GT.0) MAXF = MAX0(NFAC(KT),MAXF)
C
C COMPUTE FOURIER TRANSFORM
C
  40  DR = 8.0*FLOAT(JC)/FLOAT(KSPAN)
      CD = 2.0*SIN(0.5*DR*RAD)**2
      SD = SIN(DR*RAD)
      KK = 1
      I = I + 1
      IF (NFAC(I).NE.2) GO TO 110
C
C TRANSFORM FOR FACTOR OF 2 (INCLUDING ROTATION FACTOR)
C
      KSPAN = KSPAN/2
      K1 = KSPAN + 2
  50  K2 = KK + KSPAN
      AK = A(K2)
      BK = B(K2)
      A(K2) = A(KK) - AK
      B(K2) = B(KK) - BK
      A(KK) = A(KK) + AK
      B(KK) = B(KK) + BK
      KK = K2 + KSPAN
      IF (KK.LE.NN) GO TO 50
      KK = KK - NN
      IF (KK.LE.JC) GO TO 50
      IF (KK.GT.KSPAN) GO TO 350
  60  C1 = 1.0 - CD
      S1 = SD
      MM = MIN0(K1/2,KLIM)
      GO TO 80
  70  AK = C1 - (CD*C1+SD*S1)
      S1 = (SD*C1-CD*S1) + S1
C
C THE FOLLOWING THREE STATEMENTS COMPENSATE FOR TRUNCATION
C ERROR.  IF ROUNDED ARITHMETIC IS USED, SUBSTITUTE
C C1=AK
C
C     C1 = 0.5/(AK**2+S1**2) + 0.5
C     S1 = C1*S1
C     C1 = C1*AK
      C1=AK
  80  K2 = KK + KSPAN
      AK = A(KK) - A(K2)
      BK = B(KK) - B(K2)
      A(KK) = A(KK) + A(K2)
      B(KK) = B(KK) + B(K2)
      A(K2) = C1*AK - S1*BK
```

```
      B(K2) = S1*AK + C1*BK
      KK = K2 + KSPAN
      IF (KK.LT.NT) GO TO 80
      K2 = KK - NT
      C1 = -C1
      KK = K1 - K2
      IF (KK.GT.K2) GO TO 80
      KK = KK + JC
      IF (KK.LE.MM) GO TO 70
      IF (KK.LT.K2) GO TO 90
      K1 = K1 + INC + INC
      KK = (K1-KSPAN)/2 + JC
      IF (KK.LE.JC+JC) GO TO 60
      GO TO 40
   90 S1 = FLOAT((KK-1)/JC)*DR*RAD
      C1 = COS(S1)
      S1 = SIN(S1)
      MM = MIN0(K1/2,MM+KLIM)
      GO TO 80
C
C TRANSFORM FOR FACTOR OF 3 (OPTIONAL CODE)
C
  100 K1 = KK + KSPAN
      K2 = K1 + KSPAN
      AK = A(KK)
      BK = B(KK)
      AJ = A(K1) + A(K2)
      BJ = B(K1) + B(K2)
      A(KK) = AK + AJ
      B(KK) = BK + BJ
      AK = -0.5*AJ + AK
      BK = -0.5*BJ + BK
      AJ = (A(K1)-A(K2))*S120
      BJ = (B(K1)-B(K2))*S120
      A(K1) = AK - BJ
      B(K1) = BK + AJ
      A(K2) = AK + BJ
      B(K2) = BK - AJ
      KK = K2 + KSPAN
      IF (KK.LT.NN) GO TO 100
      KK = KK - NN
      IF (KK.LE.KSPAN) GO TO 100
      GO TO 290
C
C TRANSFORM FOR FACTOR OF 4
C
  110 IF (NFAC(I).NE.4) GO TO 230
      KSPNN = KSPAN
      KSPAN = KSPAN/4
  120 C1 = 1.0
      S1 = 0
      MM = MIN0(KSPAN,KLIM)
      GO TO 150
  130 C2 = C1 - (CD*C1+SD*S1)
```

```
      S1 = (SD*C1-CD*S1) + S1
C
C THE FOLLOWING THREE STATEMENTS COMPENSATE FOR TRUNCATION
C ERROR.  IF ROUNDED ARITHMETIC IS USED, SUBSTITUTE
C C1=C2
C
C     C1 = 0.5/(C2**2+S1**2) + 0.5
C     S1=C1*S1
C     C1 = C1*C2
      C1=C2
  140 C2 = C1**2 - S1**2
      S2 = C1*S1*2.0
      C3 = C2*C1 - S2*S1
      S3 = C2*S1 + S2*C1
  150 K1 = KK + KSPAN
      K2 = K1 + KSPAN
      K3 = K2 + KSPAN
      AKP = A(KK) + A(K2)
      AKM = A(KK) - A(K2)
      AJP = A(K1) + A(K3)
      AJM = A(K1) - A(K3)
      A(KK) = AKP + AJP
      AJP = AKP - AJP
      BKP = B(KK) + B(K2)
      BKM = B(KK) - B(K2)
      BJP = B(K1) + B(K3)
      BJM = B(K1) - B(K3)
      B(KK) = BKP + BJP
      BJP = BKP - BJP
      IF (ISN.LT.0) GO TO 180
      AKP = AKM - BJM
      AKM = AKM + BJM
      BKP = BKM + AJM
      BKM = BKM - AJM
      IF (S1.EQ.0.0) GO TO 190
  160 A(K1) = AKP*C1 - BKP*S1
      B(K1) = AKP*S1 + BKP*C1
      A(K2) = AJP*C2 - BJP*S2
      B(K2) = AJP*S2 + BJP*C2
      A(K3) = AKM*C3 - BKM*S3
      B(K3) = AKM*S3 + BKM*C3
      KK = K3 + KSPAN
      IF (KK.LE.NT) GO TO 150
  170 KK = KK - NT + JC
      IF (KK.LE.MM) GO TO 130
      IF (KK.LT.KSPAN) GO TO 200
      KK = KK - KSPAN + INC
      IF (KK.LE.JC) GO TO 120
      IF (KSPAN.EQ.JC) GO TO 350
      GO TO 40
  180 AKP = AKM + BJM
      AKM = AKM - BJM
      BKP = BKM - AJM
      BKM = BKM + AJM
```

```
      IF (S1.NE.0.0) GO TO 160
 190  A(K1) = AKP
      B(K1) = BKP
      A(K2) = AJP
      A(K3) = AKM
      B(K2) = BJP
      B(K3) = BKM
      KK = K3 + KSPAN
      IF (KK.LE.NT) GO TO 150
      GO TO 170
 200  S1 = FLOAT((KK-1)/JC)*DR*RAD
      C1 = COS(S1)
      S1 = SIN(S1)
      MM = MIN0(KSPAN,MM+KLIM)
      GO TO 140
C
C TRANSFORM FOR FACTOR OF 5 (OPTIONAL CODE)
C
 210  C2 = C72**2 - S72**2
      S2 = 2.0*C72*S72
 220  K1 = KK + KSPAN
      K2 = K1 + KSPAN
      K3 = K2 + KSPAN
      K4 = K3 + KSPAN
      AKP = A(K1) + A(K4)
      AKM = A(K1) - A(K4)
      BKP = B(K1) + B(K4)
      AJP = A(K2) + A(K3)
      AJM = A(K2) - A(K3)
      BJP = B(K2) + B(K3)
      BJM = B(K2) - B(K3)
      AA = A(KK)
      BB = B(KK)
      A(KK) = AA + AKP + AJP
      B(KK) = BB + BKP + BJP
      AK = AKP*C72 + AJP*C2 + AA
      BK = BKP*C72 + BJP*C2 + BB
      AJ = AKM*S72 + AJM*S2
      BJ = BKM*S72 + BJM*S2
      A(K1) = AK - BJ
      A(K4) = AK + BJ
      B(K1) = BK + AJ
      B(K4) = BK - AJ
      AK = AKP*C2 + AJP*C72 + AA
      BK = BKP*C2 + BJP*C72 + BB
      AJ = AKM*S2 - AJM*S72
      BJ = BKM*S2 - BJM*S72
      A(K2) = AK - BJ
      A(K3) = AK + BJ
      B(K2) = BK + AJ
      B(K3) = BK - AJ
      KK = K4 + KSPAN
      IF (KK.LT.NN) GO TO 220
      KK = KK - NN
```

```
      IF (KK.LE.KSPAN) GO TO 220
      GO TO 290
C
C TRANSFORM FOR ODD FACTORS
C
 230  K = NFAC(I)
      KSPNN = KSPAN
      KSPAN = KSPAN/K
      IF (K.EQ.3) GO TO 100
      IF (K.EQ.5) GO TO 210
      IF (K.EQ.JF) GO TO 250
      JF = K
      S1 = RAD/(FLOAT(K)/8.0)
      C1 = COS(S1)
      S1 = SIN(S1)
      CK(JF) = 1.0
      SK(JF) = 0.0
      J = 1
 240  CK(J) = CK(K)*C1 + SK(K)*S1
      SK(J) = CK(K)*S1 - SK(K)*C1
      K = K - 1
      CK(K) = CK(J)
      SK(K) = -SK(J)
      J = J + 1
      IF (J.LT.K) GO TO 240
 250  K1 = KK
      K2 = KK + KSPNN
      AA = A(KK)
      BB = B(KK)
      AK = AA
      BK = BB
      J = 1
      K1 = K1 + KSPAN
 260  K2 = K2 - KSPAN
      J = J + 1
      AT(J) = A(K1) + A(K2)
      AK = AT(J) + AK
      BT(J) = B(K1) + B(K2)
      BK = BT(J) + BK
      J = J + 1
      AT(J) = A(K1) - A(K2)
      BT(J) = B(K1) - B(K2)
      K1 = K1 + KSPAN
      IF (K1.LT.K2) GO TO 260
      A(KK) = AK
      B(KK) = BK
      K1 = KK
      K2 = KK + KSPNN
      J = 1
 270  K1 = K1 + KSPAN
      K2 = K2 - KSPAN
      JJ = J
      AK = AA
      BK = BB
```

```
      AJ = 0.0
      BJ = 0.0
      K = 1
  280 K = K + 1
      AK = AT(K)*CK(JJ) + AK
      BK = BT(K)*CK(JJ) + BK
      K = K + 1
      AJ = AT(K)*SK(JJ) + AJ
      BJ = BT(K)*SK(JJ) + BJ
      JJ = JJ + J
      IF (JJ.GT.JF) JJ = JJ - JF
      IF (K.LT.JF) GO TO 280
      K = JF - J
      A(K1) = AK - BJ
      B(K1) = BK + AJ
      A(K2) = AK + BJ
      B(K2) = BK - AJ
      J = J + 1
      IF (J.LT.K) GO TO 270
      KK = KK + KSPNN
      IF (KK.LE.NN) GO TO 250
      KK = KK - NN
      IF (KK.LE.KSPAN) GO TO 250
C
C MULTIPLY BY ROTATION FACTOR (EXCEPT FOR FACTORS OF 2 AND 4)
C
  290 IF (I.EQ.M) GO TO 350
      KK = JC + 1
  300 C2 = 1.0 - CD
      S1 = SD
      MM = MIN0(KSPAN,KLIM)
      GO TO 320
  310 C2 = C1 - (CD*C1+SD*S1)
      S1 = S1 + (SD*C1-CD*S1)
C
C THE FOLLOWING THREE STATEMENTS COMPENSATE FOR TRUNCATION
C ERROR.  IF ROUNDED ARITHMETIC IS USED, THEY MAY
C BE DELETED.
C
C     C1 = 0.5/(C2**2+S1**2) + 0.5
C     S1 = C1*S1
C     C2 = C1*C2
  320 C1 = C2
      S2 = S1
      KK = KK + KSPAN
  330 AK = A(KK)
      A(KK) = C2*AK - S2*B(KK)
      B(KK) = S2*AK + C2*B(KK)
      KK = KK + KSPNN
      IF (KK.LE.NT) GO TO 330
      AK = S1*S2
      S2 = S1*C2 + C1*S2
      C2 = C1*C2 - AK
      KK = KK - NT + KSPAN
```

```
      IF (KK.LE.KSPNN) GO TO 330
      KK = KK - KSPNN + JC
      IF (KK.LE.MM) GO TO 310
      IF (KK.LT.KSPAN) GO TO 340
      KK = KK - KSPAN + JC + INC
      IF (KK.LE.JC+JC) GO TO 300
      GO TO 40
 340  S1 = FLOAT((KK-1)/JC)*DR*RAD
      C2 = COS(S1)
      S1 = SIN(S1)
      MM = MIN0(KSPAN,MM+KLIM)
      GO TO 320
C
C PERMUTE THE RESULTS TO NORMAL ORDER---DONE IN TWO STAGES
C PERMUTATION FOR SQUARE FACTORS OF N
C
 350  NP(1) = KS
      IF (KT.EQ.0) GO TO 440
      K = KT + KT + 1
      IF (M.LT.K) K = K - 1
      J = 1
      NP(K+1) = JC
 360  NP(J+1) = NP(J)/NFAC(J)
      NP(K) = NP(K+1)*NFAC(J)
      J = J + 1
      K = K - 1
      IF (J.LT.K) GO TO 360
      K3 = NP(K+1)
      KSPAN=NP(2)
      KK = JC + 1
      K2 = KSPAN + 1
      J = 1
      IF (N.NE.NTOT) GO TO 400
C
C PERMUTATION FOR SINGLE-VARIATE TRANSFORM (OPTIONAL CODE)
C
 370  AK = A(KK)
      A(KK) = A(K2)
      A(K2) = AK
      BK = B(KK)
      B(KK) = B(K2)
      B(K2) = BK
      KK = KK + INC
      K2 = KSPAN + K2
      IF (K2.LT.KS) GO TO 370
 380  K2 = K2 - NP(J)
      J = J + 1
      K2 = NP(J+1) + K2
      IF (K2.GT.NP(J)) GO TO 380
      J = 1
 390  IF (KK.LT.K2) GO TO 370
      KK = KK + INC
      K2 = KSPAN + K2
      IF (K2.LT.KS) GO TO 390
```

```
      IF (KK.LT.KS) GO TO 380
      JC = K3
      GO TO 440
C
C PERMUTATION FOR MULTIVARIATE TRANSFORM
C
 400  K = KK + JC
 410  AK = A(KK)
      A(KK) = A(K2)
      A(K2) = AK
      BK = B(KK)
      B(KK) = B(K2)
      B(K2) = BK
      KK = KK + INC
      K2 = K2 + INC
      IF (KK.LT.K) GO TO 410
      KK = KK + KS - JC
      K2 = K2 + KS - JC
      IF (KK.LT.NT) GO TO 400
      K2 = K2 - NT + KSPAN
      KK = KK - NT + JC
      IF (K2.LT.KS) GO TO 400
 420  K2 = K2 - NP(J)
      J = J + 1
      K2 = NP(J+1) + K2
      IF (K2.GT.NP(J)) GO TO 420
      J = 1
 430  IF (KK.LT.K2) GO TO 400
      KK = KK + JC
      K2 = KSPAN + K2
      IF (K2.LT.KS) GO TO 430
      IF (KK.LT.KS) GO TO 420
      JC = K3
 440  IF (2*KT+1.GE.M) RETURN
      KSPNN = NP(KT+1)
C
C PERMUTATION FOR SQUARE-FREE FACTORS OF N
C
      J = M - KT
      NFAC(J+1) = 1
 450  NFAC(J) = NFAC(J)*NFAC(J+1)
      J = J - 1
      IF (J.NE.KT) GO TO 450
      KT = KT + 1
      NN = NFAC(KT) - 1
      JJ = 0
      J = 0
      GO TO 480
 460  JJ = JJ - K2
      K2 = KK
      K = K + 1
      KK = NFAC(K)
 470  JJ = KK + JJ
      IF (JJ.GE.K2) GO TO 460
```

```
      NP(J) = JJ
 480  K2 = NFAC(KT)
      K = KT + 1
      KK = NFAC(K)
      J = J + 1
      IF (J.LE.NN) GO TO 470
C
C DETERMINE THE PERMUTATION CYCLES OF LENGTH GREATER THAN 1
C
      J = 0
      GO TO 500
 490  K = KK
      KK = NP(K)
      NP(K) = -KK
      IF (KK.NE.J) GO TO 490
      K3 = KK
 500  J = J + 1
      KK = NP(J)
      IF (KK.LT.0) GO TO 500
      IF (KK.NE.J) GO TO 490
      NP(J) = -J
      IF (J.NE.NN) GO TO 500
      MAXF = INC*MAXF
C
C REORDER A AND B, FOLLOWING THE PERMUTATION CYCLES
C
      GO TO 570
 510  J = J - 1
      IF (NP(J).LT.0) GO TO 510
      JJ = JC
 520  KSPAN = JJ
      IF (JJ.GT.MAXF) KSPAN = MAXF
      JJ = JJ - KSPAN
      K = NP(J)
      KK = JC*K + I + JJ
      K1 = KK + KSPAN
      K2 = 0
 530  K2 = K2 + 1
      AT(K2) = A(K1)
      BT(K2) = B(K1)
      K1 = K1 - INC
      IF (K1.NE.KK) GO TO 530
 540  K1 = KK + KSPAN
      K2 = K1 - JC*(K+NP(K))
      K = -NP(K)
 550  A(K1) = A(K2)
      B(K1) = B(K2)
      K1 = K1 - INC
      K2 = K2 - INC
      IF (K1.NE.KK) GO TO 550
      KK = K2
      IF (K.NE.J) GO TO 540
      K1 = KK + KSPAN
      K2 = 0
```

```
560  K2 = K2 + 1
     A(K1) = AT(K2)
     B(K1) = BT(K2)
     K1 = K1 - INC
     IF (K1.NE.KK) GO TO 560
     IF (JJ.NE.0) GO TU 520
     IF (J.NE.1) GO TO 510
570  J = K3 + 1
     NT = NT - KSPNN
     I = NT - INC + 1
     IF (NT.GE.0) GO TO 510
     RETURN
     END
```

APPENDIX C

Winograd Fourier Transform Algorithm Program

This appendix contains the driver and algorithm subroutine for executing and timing the Winograd Fourier Transform Algorithm (WFTA). The program was developed by McClellan and Nawab (McClellan and Nawab, 1979). The program consists of six subroutines called in the following order.

INISHL -INISHL is called to initialize the algorithm. It decomposes N into four relatively prime factors, computes the multiplication coefficients, and generates the indices for the input and output permutations. INISHL needs to be performed only when a new sequence length is to be transformed.

PERM1 -PERM1 permutes the input sequence in arrays XR and XI and stores the reordered sequence in arrays SR and SI.

WEAVE1 -WEAVE1 performs the input additions for the short transforms. WEAVE1 contains a module for each factor's short transform.

MULT -MULT performs the nested multiplications.

WEAVE2 -WEAVE2 performs the output additions.

PERM2 -PERM2 permutes the coefficients in arrays SR and SI and stores them in arrays XR and XI.

The variables passed are defined as follows.

N -N is an integer variable specifying the length of the input sequence x(n) and the output sequence X(k).

XR -XR is a real array storing the real components of the original sequence and final output sequence. It must have length of at least N.

XI -XI is a real array of dimension N. It stores the imaginary components of the input and output sequence.

INIT -INIT is an integer value which, if set to zero, indicates that INISHL should be called to perform the initialization.

IERR -IERR is an integer value returned by the subroutine to indicate an error condition. It can have the following values:

IERR=0: no errors

IERR=-1: N cannot be factored properly

IERR=-2: not initialized for this value of N.

SR -SR is a real array of length (ND1)(ND2)(ND3)(ND4). It stores the real component of the intermediate results.

SI -SI is a real array of length (ND1)(ND2)(ND3)(ND4). It stores the imaginary component of the intermediate results.

```
      DIMENSION INDX1(504),INDX2(504)
      DIMENSION SR(792),SI(792),COEF(792)
      DIMENSION XR(504),XI(504)
      DIMENSION XAVG(4)
      READ(5,*) NUMTIM
      DO 999 INUM=1,NUMTIM
      READ(5,*) T,DELT,M,LIST
      WRITE(6,601) T,DELT,M
601   FORMAT(1H1,'TEST WITH T =',F10.2,' DELT = ',F10.2,'MULTIPLES',I5)
      LCOUNT = 0
      K = 0
111   CONTINUE
      LCOUNT = LCOUNT + 1
      INVRS = 0
      INIT = 0
101   CONTINUE
      K = K + 1
      IF(K.GT.1) INIT = 1
      N = (T/DELT) + 0.4
      DELF = 1.0/T
      PI = 3.141592653898
      E = 2.71828
      FO = 25.0
      DO 10 I=1,N
        T1 = (I-1)*DELT
        ARG = 2.0*PI*FO*T1
        TE = -1.0*T1
        XI(I) = 0.0
10      XR(I) = (E**TE)*COS(ARG)
      CALL WFTA(N,XR,XI,INIT,IERR,SR,SI,COEF,M,INDX1,INDX2,XOUT)
      IF(IERR.NE.0) GO TO 900
      PRINT*,'XOUT',XOUT
      XAVG(LCOUNT) = XOUT
      DO 30 I=1,NPTS
30      XR(I) = ((XR(I)**2) + (XI(I)**2)) ** 0.5
      PRINT*, N
      IF(LCOUNT.LT.3) GO TO 111
      TAVG = 0.
      DO 599 J=1,3
        TAVG = TAVG + XAVG(J)
599     CONTINUE
      TAVG = TAVG/3
      PRINT*, 'AVG XOUT = ',TAVG
      IF(K.LE.1) GO TO 101
      IF(LIST.NE.1) GO TO 999
      WRITE(6,600)
600   FORMAT('1',20X,'MAG',16X,'IMAG',16X,16X,/)
      WRITE(6,700) (I,XR(I),XI(I),I=1,N)
700   FORMAT(5X,I5,5X,E14.7,5X,E14.7,5X)
999   CONTINUE
      GO TO 998
900   PRINT*,'IERR = ',IERR
998   STOP
      END
```

```
      SUBROUTINE WFTA(N,XR,XI,INIT,IERR,SR,SI,COEF,M,INDX1,INDX2,XOUT)
      REAL    XR(1),XI(1)
C
C  ********************************************
C     THE FOLLOWING TWO CARDS MAY BE CHANGED IF THE MAXIMUM
C     DESIRED TRANSFORM LENGTH IS LESS THAN 5040
      REAL SR(1),SI(1),COEF(1)
      INTEGER INDX1(1),INDX2(1)
C  ********************************************
C
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
C
C       TEST FOR INITIAL RUN
      XIN = SECNDS(0.0)
      IF(INIT.EQ.0) CALL INISHL(N,COEF,XR,XI,INDX1,INDX2,IERR)
C
      IF(IERR.LT.0) RETURN
      M=NA*NB*NC*ND
      IF(M.EQ.N) GO TO 100
      IERR=-2
      RETURN
C
C    PROGRAM NOT INITIALIZED FOR THIS VALUE OF N
C
100   NMULT=ND1*ND2*ND3*ND4
C
C       PERMUTE THE INPUT DATA
C
      CALL PERM1(SR,SI,XR,XI,INDX1)
C
C       DO THE PRE-WEAVE MODULES
C
      CALL WEAVE1(SR,SI)
C
C       DO THE NESTED MULTIPLICATIONS
C
      CALL MULT(SR,SI,COEF,NMULT)
C
C       DO THE POST-WEAVE MODULES
C
      CALL WEAVE2(SR,SI)
C
C       PERMUTE THE OUTPUT DATA
C
      CALL PERM2(SR,SI,XR,XI,INDX2)
      XOUT = SECNDS(XIN)
      RETURN
      END
      SUBROUTINE INISHL(N,COEF,XR,XI,INDX1,INDX2,IERR)
      REAL    COEF(1),XR(1),XI(1)
      INTEGER S1,S2,S3,S4,INDX1(1),INDX2(1),P1
      REAL    CO3(3),CO4(4),CO8(8),CO9(11),CO16(18),CD1(18),CD2(11),
     1CD3(9),CD4(6)
```

```
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
C
C      DATA STATEMENTS ASSIGN SHORT DFT COEFFICIENTS.
C
      DATA CO3/1.0,-1.5,-0.8660254038/
      DATA CO4/1.0,1.0,1.0,1.0/
      DATA CO8/1.0,1.0,1.0,-1.0,1.0,-0.7071067812,
     1 -1.0,0.7071067812/
      DATA CO9/1.0,-1.5,-0.8660254038,0.5,0.7660444431,
     1 -0.1736481777,0.9396926208,-0.6427876097,
     2 -0.984807753,0.3420201433,-0.8660254038/
      DATA CO16/1.0,1.0,1.0,-1.0,1.0,-0.7071067812,-1.0,
     1 0.7071067812,1.0,0.5411961001,-0.7071067812,
     2 -0.5411961001,-1.0,-1.306562965,0.7071067812,
     3 1.306562965,-0.9238795325,0.3826834324/
      DATA CD1/18*1./
      DATA CD2/11*1./
      DATA CD3/1.0,-1.1666666667,-0.4409585518,0.7343022012,
     1 0.7901564685,-0.3408729306,-0.874842291,
     2 0.0558542673,0.5339693603/
      DATA CD4/1.0,-1.25,-1.538841769,0.5590169944,0.363271264,
     1 0.5877852523/
C
C      FOLLOWING SEGMENT DETERMINES FACTORS OF N AND CHOOSES
C      THE APPROPRIATE SHORT DFT COEFFICIENTS.
C
      IERR=0
      ND1=1
      NA=1
      NB=1
      ND2=1
      NC=1
      ND3=1
      ND=1
      ND4=1
      IF(N.LE.0) GO TO 190
      IF(16*(N/16).EQ.N) GO TO 30
      IF(8*(N/8).EQ.N) GO TO 40
      IF(4*(N/4).EQ.N) GO TO 50
      IF(2*(N/2).NE.N) GO TO 70
      ND1=2
      NA=2
      CD1(2)=1.0
      GO TO 70
30    ND1=18
      NA=16
      DO 31 J=1,18
31    CD1(J)=CO16(J)
      GO TO 70
40    ND1=8
      NA=8
      DO 41 J=1,8
41    CD1(J)=CO8(J)
      GO TO 70
```

```
50      ND1=4
        NA=4
        DO 51 J=1,4
51      CD1(J)=CO4(J)
70      IF(3*(N/3).NE.N) GO TO 120
        IF(9*(N/9).EQ.N) GO TO 100
        ND2=3
        NB=3
        DO 71 J=1,3
71      CD2(J)=CO3(J)
        GO TO 120
100     ND2=11
        NB=9
        DO 110 J=1,11
110     CD2(J)=CO9(J)
120     IF(7*(N/7).NE.N) GO TO 160
        ND3=9
        NC=7
160     IF(5*(N/5).NE.N) GO TO 190
        ND4=6
        ND=5
190     M=NA*NB*NC*ND
        IF(M.EQ.N) GO TO 250
210     PRINT*,'THIS N DOES NOT WORK'
        IERR=-1
        RETURN
C
C        NEXT SEGMENT GENERATES THE DFT COEFFICIENTS AND
C        THE FLAG ARRAY.
250     J=1
        DO 300 N4=1,ND4
        DO 300 N3=1,ND3
        DO 300 N2=1,ND2
        DO 300 N1=1,ND1
        COEF(J)=CD1(N1)*CD2(N2)*CD3(N3)*CD4(N4)
        J=J+1
300     CONTINUE
C        FOLLOWING SEGMENT FOR INPUT INDEXING.
        S1=0
        S2=0
        S3=0
        S4=0
        S5=0
        J=1
        NU=NB*NC*ND
        NV=NA*NC*ND
        NW=NA*NB*ND
        NY=NA*NB*NC
        K=1
        DO 440 N4=1,ND
        DO 430 N3=1,NC
        DO 420 N2=1,NB
        DO 410 N1=1,NA
405     IF(K.LE.N) GO TO 408
```

```
      K=K-N
      GO TO 405
408   INDX1(J)=K
      J=J+1
410   K=K+NU
420   K=K+NV
430   K=K+NW
440   K=K+NY
C
C      FOLLOWING SEGMENT FOR OUTPUT INDEXING
      M=1
      IF(NA.EQ.1) GO TO 530
520   P1=M*NU-1
      IF((P1/NA)*NA.EQ.P1) GO TO 510
      M=M+1
      GO TO 520
510   S1=P1+1
530   IF(NB.EQ.1) GO TO 540
      M=1
550   P1=M*NV-1
      IF((P1/NB)*NB.EQ.P1) GO TO 560
      M=M+1
      GO TO 550
560   S2=P1+1
540   IF(NC.EQ.1) GO TO 630
      M=1
620   P1=M*NW-1
      IF((P1/NC)*NC.EQ.P1) GO TO 610
      M=M+1
      GO TO 620
610   S3=P1+1
630   IF(ND.EQ.1) GO TO 660
      M=1
640   P1=M*NY-1
      IF((P1/ND)*ND.EQ.P1) GO TO 650
      M=M+1
      GO TO 640
650   S4=P1+1
660   J=1
      DO 810 N4=1,ND
      DO 810 N3=1,NC
      DO 810 N2=1,NB
      DO 810 N1=1,NA
      INDX2(J)=S1*(N1-1)+S2*(N2-1)+S3*(N3-1)+S4*(N4-1)+1
900   IF(INDX2(J).LE.N) GO TO 910
      INDX2(J)=INDX2(J)-N
      GO TO 900
910   J=J+1
810   CONTINUE
      RETURN
      END
      SUBROUTINE PERM1(SR,SI,XR,XI,INDX1)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL   XR(1),XI(1),SR(1),SI(1)
```

```
      INTEGER INDX1(1)
      J=1
      K=1
      INC1=ND1-NA
      INC2=ND1*(ND2-NB)
      INC3=ND1*ND2*(ND3-NC)
      DO 40 N4=1,ND
      DO 30 N3=1,NC
      DO 20 N2=1,NB
      DO 10 N1=1,NA
      SR(J)=XR(INDX1(K))
      SI(J)=XI(INDX1(K))
      K=K+1
10    J=J+1
20    J=J+INC1
30    J=J+INC2
40    J=J+INC3
      RETURN
      END
      SUBROUTINE PERM2(SR,SI,XR,XI,INDX2)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL   XR(1),XI(1),SR(1),SI(1)
      INTEGER INDX2(1)
      J=1
      K=1
      INC1=ND1-NA
      INC2=ND1*(ND2-NB)
      INC3=ND1*ND2*(ND3-NC)
      DO 40 N4=1,ND
      DO 30 N3=1,NC
      DO 20 N2=1,NB
      DO 10 N1=1,NA
      XR(INDX2(K))=SR(J)
      XI(INDX2(K))=SI(J)
      K=K+1
10    J=J+1
20    J=J+INC1
30    J=J+INC2
40    J=J+INC3
      RETURN
      END
      SUBROUTINE WEAVE1(SR,SI)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL Q(8),T(16),SR(1),SI(1)
      IF(NA.EQ.1) GO TO 300
      IF(NA.NE.2) GO TO 800
C
C ***************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 2 POINT PRE-WEAVE MODULE
C
C ***************************************************************
C
      NLUP2=2*(ND2-NB)
```

```
      NLUP23=2*ND2*(ND3-NC)
      NBASE=1
      DO 240 N4=1,ND
      DO 230 N3=1,NC
      DO 220 N2=1,NB
      NR1=NBASE+1
      T0=SR(NBASE)+SR(NR1)
      SR(NR1)=SR(NBASE)-SR(NR1)
      SR(NBASE)=T0
      T0=SI(NBASE)+SI(NR1)
      SI(NR1)=SI(NBASE)-SI(NR1)
      SI(NBASE)=T0
220   NBASE=NBASE+2
230   NBASE=NBASE+NLUP2
240   NBASE=NBASE+NLUP23
800   IF(NA.NE.8) GO TO 1600
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 8 POINT PRE-WEAVE MODULE
C
C ****************************************************************
C
      NLUP2=8*(ND2-NB)
      NLUP23=8*ND2*(ND3-NC)
      NBASE=1
      DO 840 N4=1,ND
      DO 830 N3=1,NC
      DO 820 N2=1,NB
      NR1=NBASE+1
      NR2=NR1+1
      NR3=NR2+1
      NR4=NR3+1
      NR5=NR4+1
      NR6=NR5+1
      NR7=NR6+1
      T3=SR(NR3)+SR(NR7)
      T7=SR(NR3)-SR(NR7)
      T0=SR(NBASE)+SR(NR4)
      SR(NR4)=SR(NBASE)-SR(NR4)
      T1=SR(NR1)+SR(NR5)
      T5=SR(NR1)-SR(NR5)
      T2=SR(NR2)+SR(NR6)
      SR(NR6)=SR(NR2)-SR(NR6)
      SR(NBASE)=T0+T2
      SR(NR2)=T0-T2
      SR(NR1)=T1+T3
      SR(NR3)=T1-T3
      SR(NR5)=T5+T7
      SR(NR7)=T5-T7
      T3=SI(NR3)+SI(NR7)
      T7=SI(NR3)-SI(NR7)
      T0=SI(NBASE)+SI(NR4)
      SI(NR4)=SI(NBASE)-SI(NR4)
```

```
      T1=SI(NR1)+SI(NR5)
      T5=SI(NR1)-SI(NR5)
      T2=SI(NR2)+SI(NR6)
      SI(NR6)=SI(NR2)-SI(NR6)
      SI(NBASE)=T0+T2
      SI(NR2)=T0-T2
      SI(NR1)=T1+T3
      SI(NR3)=T1-T3
      SI(NR5)=T5+T7
      SI(NR7)=T5-T7
820   NBASE=NBASE+8
830   NBASE=NBASE+NLUP2
840   NBASE=NBASE+NLUP23
1600  IF(NA.NE.16) GO TO 300
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 16 POINT PRE-WEAVE MODULE
C
C ****************************************************************
C
      NLUP2=16*(ND2-NB)
      NLUP23=16*ND2*(ND3-NC)
      NBASE=1
      DO 1640 N4=1,ND
      DO 1630 N3=1,NC
      DO 1620 N2=1,NB
      NR1=NBASE+1
      NR2=NR1+1
      NR3=NR2+1
      NR4=NR3+1
      NR5=NR4+1
      NR6=NR5+1
      NR7=NR6+1
      NR8=NR7+1
      NR9=NR8+1
      NR10=NR9+1
      NR11=NR10+1
      NR12=NR11+1
      NR13=NR12+1
      NR14=NR13+1
      NR15=NR14+1
      NR16=NR15+1
      NR17=NR16+1
      JBASE=NBASE
      DO 1645 J=1,8
      T(J)=SR(JBASE)+SR(JBASE+8)
      T(J+8)=SR(JBASE)-SR(JBASE+8)
      JBASE=JBASE+1
1645  CONTINUE
      DO 1650 J=1,4
      Q(J)=T(J)+T(J+4)
      Q(J+4)=T(J)-T(J+4)
1650  CONTINUE
```

```
      SR(NBASE)=Q(1)+Q(3)
      SR(NR2)=Q(1)-Q(3)
      SR(NR1)=Q(2)+Q(4)
      SR(NR3)=Q(2)-Q(4)
      SR(NR5)=Q(6)+Q(8)
      SR(NR7)=Q(6)-Q(8)
      SR(NR4)=Q(5)
      SR(NR6)=Q(7)
      SR(NR8)=T(9)
      SR(NR9)=T(10)+T(16)
      SR(NR15)=T(10)-T(16)
      SR(NR13)=T(14)+T(12)
      SR(NR11)=T(14)-T(12)
      SR(NR17)=SR(NR11)+SR(NR15)
      SR(NR16)=SR(NR9)+SR(NR13)
      SR(NR10)=T(11)+T(15)
      SR(NR14)=T(11)-T(15)
      SR(NR12)=T(13)
      JBASE=NBASE
      DO 1745 J=1,8
      T(J)=SI(JBASE)+SI(JBASE+8)
      T(J+8)=SI(JBASE)-SI(JBASE+8)
      JBASE=JBASE+1
1745  CONTINUE
      DO 1750 J=1,4
      Q(J)=T(J)+T(J+4)
      Q(J+4)=T(J)-T(J+4)
1750  CONTINUE
      SI(NBASE)=Q(1)+Q(3)
      SI(NR2)=Q(1)-Q(3)
      SI(NR1)=Q(2)+Q(4)
      SI(NR3)=Q(2)-Q(4)
      SI(NR5)=Q(6)+Q(8)
      SI(NR7)=Q(6)-Q(8)
      SI(NR4)=Q(5)
      SI(NR6)=Q(7)
      SI(NR8)=T(9)
      SI(NR9)=T(10)+T(16)
      SI(NR15)=T(10)-T(16)
      SI(NR13)=T(14)+T(12)
      SI(NR11)=T(14)-T(12)
      SI(NR17)=SI(NR11)+SI(NR15)
      SI(NR16)=SI(NR9)+SI(NR13)
      SI(NR10)=T(11)+T(15)
      SI(NR14)=T(11)-T(15)
      SI(NR12)=T(13)
1620  NBASE=NBASE+18
1630  NBASE=NBASE+NLUP2
1640  NBASE=NBASE+NLUP23
300   IF(NB.EQ.1) GO TO 700
      IF(NB.NE.3) GO TO 900
C
C ***************************************************************
C
```

```
C     THE FOLLOWING CODE IMPLEMENTS THE 3 POINT PRE-WEAVE MODULE
C
C ****************************************************************
C
      NLUP2=2*ND1
      NLUP23=3*ND1*(ND3-NC)
      NBASE=1
      NOFF=ND1
      DO 340 N4=1,ND
      DO 330 N3=1,NC
      DO 310 N2=1,ND1
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      T1=SR(NR1)+SR(NR2)
      SR(NBASE)=SR(NBASE)+T1
      SR(NR2)=SR(NR1)-SR(NR2)
      SR(NR1)=T1
      T1=SI(NR1)+SI(NR2)
      SI(NBASE)=SI(NBASE)+T1
      SI(NR2)=SI(NR1)-SI(NR2)
      SI(NR1)=T1
310   NBASE=NBASE+1
330   NBASE=NBASE+NLUP2
340   NBASE=NBASE+NLUP23
900   IF(NB.NE.9) GO TO 700
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 9 POINT PRE-WEAVE MODULE
C
C ****************************************************************
C
      NLUP2=10*ND1
      NLUP23=11*ND1*(ND3-NC)
      NBASE=1
      NOFF=ND1
      DO 940 N4=1,ND
      DO 930 N3=1,NC
      DO 910 N2=1,ND1
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      NR9=NR8+NOFF
      NR10=NR9+NOFF
      T3=SR(NR3)+SR(NR6)
      T6=SR(NR3)-SR(NR6)
      SR(NBASE)=SR(NBASE)+T3
      T7=SR(NR7)+SR(NR2)
      T2=SR(NR7)-SR(NR2)
```

```
      SR(NR2)=T6
      T1=SR(NR1)+SR(NR8)
      T8=SR(NR1)-SR(NR8)
      SR(NR1)=T3
      T4=SR(NR4)+SR(NR5)
      T5=SR(NR4)-SR(NR5)
      SR(NR3)=T1+T4+T7
      SR(NR4)=T1-T7
      SR(NR5)=T4-T1
      SR(NR6)=T7-T4
      SR(NR10)=T2+T5+T8
      SR(NR7)=T8-T2
      SR(NR8)=T5-T8
      SR(NR9)=T2-T5
      T3=SI(NR3)+SI(NR6)
      T6=SI(NR3)-SI(NR6)
      SI(NBASE)=SI(NBASE)+T3
      T7=SI(NR7)+SI(NR2)
      T2=SI(NR7)-SI(NR2)
      SI(NR2)=T6
      T1=SI(NR1)+SI(NR8)
      T8=SI(NR1)-SI(NR8)
      SI(NR1)=T3
      T4=SI(NR4)+SI(NR5)
      T5=SI(NR4)-SI(NR5)
      SI(NR3)=T1+T4+T7
      SI(NR4)=T1-T7
      SI(NR5)=T4-T1
      SI(NR6)=T7-T4
      SI(NR10)=T2+T5+T8
      SI(NR7)=T8-T2
      SI(NR8)=T5-T8
      SI(NR9)=T2-T5
910   NBASE=NBASE+1
930   NBASE=NBASE+NLUP2
940   NBASE=NBASE+NLUP23
700   IF(NC.NE.7) GO TO 500
C
C ***************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 7 POINT PRE-WEAVE MODULE
C ***************************************************************
C
      NOFF=ND1*ND2
      NBASE=1
      NLUP2=8*NOFF
      DO 740 N4=1,ND
      DO 710 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
```

```
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      T1=SR(NR1)+SR(NR6)
      T6=SR(NR1)-SR(NR6)
      T4=SR(NR4)+SR(NR3)
      T3=SR(NR4)-SR(NR3)
      T2=SR(NR2)+SR(NR5)
      T5=SR(NR2)-SR(NR5)
      SR(NR5)=T6-T3
      SR(NR2)=T5+T3+T6
      SR(NR6)=T5-T6
      SR(NR8)=T3-T5
      SR(NR3)=T2-T1
      SR(NR4)=T1-T4
      SR(NR7)=T4-T2
      T1=T1+T4+T2
      SR(NBASE)=SR(NBASE)+T1
      SR(NR1)=T1
      T1=SI(NR1)+SI(NR6)
      T6=SI(NR1)-SI(NR6)
      T4=SI(NR4)+SI(NR3)
      T3=SI(NR4)-SI(NR3)
      T2=SI(NR2)+SI(NR5)
      T5=SI(NR2)-SI(NR5)
      SI(NR5)=T6-T3
      SI(NR2)=T5+T3+T6
      SI(NR6)=T5-T6
      SI(NR8)=T3-T5
      SI(NR3)=T2-T1
      SI(NR4)=T1-T4
      SI(NR7)=T4-T2
      T1=T1+T4+T2
      SI(NBASE)=SI(NBASE)+T1
      SI(NR1)=T1
710   NBASE=NBASE+1
740   NBASE=NBASE+NLUP2
500   IF(ND.NE.5) RETURN
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 5 POINT PRE-WEAVE MODULE
C
C ****************************************************************
C
      NOFF=ND1*ND2*ND3
      NBASE=1
      DO 510 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      T4=SR(NR1)-SR(NR4)
      T1=SR(NR1)+SR(NR4)
```

```
      T3=SR(NR3)+SR(NR2)
      T2=SR(NR3)-SR(NR2)
      SR(NR3)=T1-T3
      SR(NR1)=T1+T3
      SR(NBASE)=SR(NBASE)+SR(NR1)
      SR(NR5)=T2+T4
      SR(NR2)=T4
      SR(NR4)=T2
      T4=SI(NR1)-SI(NR4)
      T1=SI(NR1)+SI(NR4)
      T3=SI(NR3)+SI(NR2)
      T2=SI(NR3)-SI(NR2)
      SI(NR3)=T1-T3
      SI(NR1)=T1+T3
      SI(NBASE)=SI(NBASE)+SI(NR1)
      SI(NR5)=T2+T4
      SI(NR2)=T4
      SI(NR4)=T2
510   NBASE=NBASE+1
      RETURN
      END
      SUBROUTINE MULT (SR,SI,COEF,NMULT)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL   SR(1),SI(1),COEF(1)
      DO 10 J=1,NMULT
      SR(J)=SR(J)*COEF(J)
      SI(J)=SI(J)*COEF(J)
   10 CONTINUE
      RETURN
      END
      SUBROUTINE WEAVE2(SR,SI)
      REAL   SR(1),SI(1)
      COMMON NA,NB,NC,ND,ND1,ND2,ND3,ND4
      REAL   Q(8),T(16)
      IF(ND.NE.5) GO TO 700
C
C ***************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 5 POINT POST-WEAVE MODULE
C
C ***************************************************************
C
      NOFF=ND1*ND2*ND3
      NBASE=1
      DO 510 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      T1=SR(NBASE)+SR(NR1)
      T3=T1-SR(NR3)
      T1=T1+SR(NR3)
      T4=SI(NR2)+SI(NR5)
```

```
      T2=SI(NR4)+SI(NR5)
      SR(NR1)=T1-T4
      SR4=T1+T4
      SR2=T3+T2
      SR(NR3)=T3-T2
      T1=SI(NBASE)+SI(NR1)
      T3=T1-SI(NR3)
      T1=T1+SI(NR3)
      T4=SR(NR2)+SR(NR5)
      T2=SR(NR4)+SR(NR5)
      SI(NR1)=T1+T4
      SI(NR4)=T1-T4
      SI(NR2)=T3-T2
      SI(NR3)=T3+T2
      SR(NR2)=SR2
      SR(NR4)=SR4
510   NBASE=NBASE+1
700   IF(NC.NE.7) GO TO 300
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 7 POINT POST-WEAVE MODULE
C
C ****************************************************************
C
      NOFF=ND1*ND2
      NBASE=1
      NLUP2=8*NOFF
      DO 740 N4=1,ND
      DO 710 N1=1,NOFF
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      T1=SR(NR1)+SR(NBASE)
      T2=T1-SR(NR3)-SR(NR4)
      T4=T1+SR(NR3)-SR(NR7)
      T1=T1+SR(NR4)+SR(NR7)
      T6=SI(NR2)+SI(NR5)+SI(NR8)
      T5=SI(NR2)-SI(NR5)-SI(NR6)
      T3=SI(NR2)+SI(NR6)-SI(NR8)
      SR(NR1)=T1-T6
      SR6=T1+T6
      SR2=T2-T5
      SR5=T2+T5
      SR(NR4)=T4-T3
      SR(NR3)=T4+T3
      T1=SI(NR1)+SI(NBASE)
      T2=T1-SI(NR3)-SI(NR4)
      T4=T1+SI(NR3)-SI(NR7)
```

```
      T1=T1+SI(NR4)+SI(NR7)
      T6=SR(NR2)+SR(NR5)+SR(NR8)
      T5=SR(NR2)-SR(NR5)-SR(NR6)
      T3=SR(NR2)+SR(NR6)-SR(NR8)
      SI(NR1)=T1+T6
      SI(NR6)=T1-T6
      SI(NR2)=T2+T5
      SI(NR5)=T2-T5
      SI(NR4)=T4+T3
      SI(NR3)=T4-T3
      SR(NR2)=SR2
      SR(NR5)=SR5
      SR(NR6)=SR6
710   NBASE=NBASE+1
740   NBASE=NBASE+NLUP2
300   IF(NB.EQ.1) GO TO 400
      IF(NB.NE.3) GO TO 900
C
C ******************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 3 POINT POST-WEAVE MODULE
C
C ******************************************************************
C
      NLUP2=2*ND1
      NLUP23=3*ND1*(ND3-NC)
      NBASE=1
      NOFF=ND1
      DO 340 N5=1,ND
      DO 330 N4=1,NC
      DO 310 N2=1,ND1
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      T1=SR(NBASE)+SR(NR1)
      SR(NR1)=T1-SI(NR2)
      SR2=T1+SI(NR2)
      T1=SI(NBASE)+SI(NR1)
      SI(NR1)=T1+SR(NR2)
      SI(NR2)=T1-SR(NR2)
      SR(NR2)=SR2
310   NBASE=NBASE+1
330   NBASE=NBASE+NLUP2
340   NBASE=NBASE+NLUP23
900   IF(NB.NE.9) GO TO 400
C
C ******************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 9 POINT POST-WEAVE MODULE
C
C ******************************************************************
C
      NLUP2=10*ND1
      NLUP23=11*ND1*(ND3-NC)
      NBASE=1
```

```
      NOFF=ND1
      DO 940 N4=1,ND
      DO 930 N3=1,NC
      DO 910 N2=1,ND1
      NR1=NBASE+NOFF
      NR2=NR1+NOFF
      NR3=NR2+NOFF
      NR4=NR3+NOFF
      NR5=NR4+NOFF
      NR6=NR5+NOFF
      NR7=NR6+NOFF
      NR8=NR7+NOFF
      NR9=NR8+NOFF
      NR10=NR9+NOFF
      T3=SR(NBASE)-SR(NR3)
      T7=SR(NBASE)+SR(NR1)
      SR(NBASE)=SR(NBASE)+SR(NR3)+SR(NR3)
      T6=T3+SI(NR10)
      SR(NR3)=T3-SI(NR10)
      T4=T7+SR(NR5)-SR(NR6)
      T1=T7-SR(NR4)-SR(NR5)
      T7=T7+SR(NR4)+SR(NR6)
      SR(NR6)=T6
      T8=SI(NR2)-SI(NR7)-SI(NR8)
      T5=SI(NR2)+SI(NR8)-SI(NR9)
      T2=SI(NR2)+SI(NR7)+SI(NR9)
      SR(NR1)=T7-T2
      SR8=T7+T2
      SR(NR4)=T1-T8
      SR(NR5)=T1+T8
      SR7=T4-T5
      SR2=T4+T5
      T3=SI(NBASE)-SI(NR3)
      T7=SI(NBASE)+SI(NR1)
      SI(NBASE)=SI(NBASE)+SI(NR3)+SI(NR3)
      T6=T3-SR(NR10)
      SI(NR3)=T3+SR(NR10)
      T4=T7+SI(NR5)-SI(NR6)
      T1=T7-SI(NR4)-SI(NR5)
      T7=T7+SI(NR4)+SI(NR6)
      SI(NR6)=T6
      T8=SR(NR2)-SR(NR7)-SR(NR8)
      T5=SR(NR2)+SR(NR8)-SR(NR9)
      T2=SR(NR2)+SR(NR7)+SR(NR9)
      SI(NR1)=T7+T2
      SI(NR8)=T7-T2
      SI(NR4)=T1+T8
      SI(NR5)=T1-T8
      SI(NR7)=T4+T5
      SI(NR2)=T4-T5
      SR(NR2)=SR2
      SR(NR7)=SR7
      SR(NR8)=SR8
910   NBASE=NBASE+1
```

```
930   NBASE=NBASE+NLUP2
940   NBASE=NBASE+NLUP23
400   IF(NA.EQ.1) RETURN
      IF(NA.NE.4) GO TO 800
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 4 POINT POST-WEAVE MODULE
C
C ****************************************************************
C
      NLUP2=4*(ND2-NB)
      NLUP23=4*ND2*(ND3-NC)
      NBASE=1
      DO 440 N4=1,ND
      DO 430 N3=1,NC
      DO 420 N2=1,NB
      NR1=NBASE+1
      NR2=NR1+1
      NR3=NR2+1
      TR0=SR(NBASE)+SR(NR2)
      TR2=SR(NBASE)-SR(NR2)
      TR1=SR(NR1)+SR(NR3)
      TR3=SR(NR1)-SR(NR3)
      TI1=SI(NR1)+SI(NR3)
      TI3=SI(NR1)-SI(NR3)
      SR(NBASE)=TR0+TR1
      SR(NR2)=TR0-TR1
      SR(NR1)=TR2+TI3
      SR(NR3)=TR2-TI3
      TI0=SI(NBASE)+SI(NR2)
      TI2=SI(NBASE)-SI(NR2)
      SI(NBASE)=TI0+TI1
      SI(NR2)=TI0-TI1
      SI(NR1)=TI2-TR3
      SI(NR3)=TI2+TR3
420   NBASE=NBASE+4
430   NBASE=NBASE+NLUP2
440   NBASE=NBASE+NLUP23
800   IF(NA.NE.6) GO TO 1600
C
C ****************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 6 POINT POST-WEAVE MODULE
C
C ****************************************************************
C
      NLUP2=8*(ND2-NB)
      NLUP23=8*ND2*(ND3-NC)
      NBASE=1
      DO 840 N4=1,ND
      DO 830 N3=1,NC
      DO 820 N2=1,NB
      NR1=NBASE+1
```

```
      NR2=NR1+1
      NR3=NR2+1
      NR4=NR3+1
      NR5=NR4+1
      NR6=NR5+1
      NR7=NR6+1
      T1=SR(NBASE)-SR(NR1)
      SR(NBASE)=SR(NBASE)+SR(NR1)
      SR6=SR(NR2)+SI(NR3)
      SR(NR2)=SR(NR2)-SI(NR3)
      T4=SR(NR4)-SI(NR5)
      T5=SR(NR4)+SI(NR5)
      T6=SR(NR7)-SI(NR6)
      T7=SR(NR7)+SI(NR6)
      SR(NR4)=T1
      SR(NR1)=T4+T6
      SR3=T4-T6
      SR5=T5-T7
      SR(NR7)=T5+T7
      T1=SI(NBASE)-SI(NR1)
      SI(NBASE)=SI(NBASE)+SI(NR1)
      T3=SI(NR2)-SR(NR3)
      SI(NR2)=SI(NR2)+SR(NR3)
      T4=SI(NR4)+SR(NR5)
      T5=SI(NR4)-SR(NR5)
      SI(NR6)=T3
      T6=SR(NR6)+SI(NR7)
      T7=SR(NR6)-SI(NR7)
      SI(NR4)=T1
      SI(NR1)=T4+T6
      SI(NR3)=T4-T6
      SI(NR5)=T5+T7
      SI(NR7)=T5-T7
      SR(NR3)=SR3
      SR(NR5)=SR5
      SR(NR6)=SR6
820   NBASE=NBASE+8
830   NBASE=NBASE+NLUP2
840   NBASE=NBASE+NLUP23
1600  IF(NA.NE.16) RETURN
C
C ******************************************************************
C
C     THE FOLLOWING CODE IMPLEMENTS THE 16 POINT POST-WEAVE MODULE
C
C ******************************************************************
C
      NLUP2=18*(ND2-NB)
      NLUP23=18*ND2*(ND3-NC)
      NBASE=1
      DO 1640 N4=1,ND
      DO 1630 N3=1,NC
      DO 1620 N2=1,NB
      NR1=NBASE+1
```

```
NR2=NR1+1
NR3=NR2+1
NR4=NR3+1
NR5=NR4+1
NR6=NR5+1
NR7=NR6+1
NR8=NR7+1
NR9=NR8+1
NR10=NR9+1
NR11=NR10+1
NR12=NR11+1
NR13=NR12+1
NR14=NR13+1
NR15=NR14+1
NR16=NR15+1
NR17=NR16+1
T(2)=SR(NBASE)-SR(NR1)
SR(NBASE)=SR(NR1)+SR(NBASE)
T(4)=SR(NR2)+SI(NR3)
T(3)=SR(NR2)-SI(NR3)
T(6)=SR(NR4)+SI(NR5)
T(5)=SR(NR4)-SI(NR5)
T(8)=-SI(NR6)-SR(NR7)
T(7)=-SI(NR6)+SR(NR7)
T(9)=SR(NR8)+SR(NR14)
T(15)=SR(NR8)-SR(NR14)
T(13)=-SI(NR10)-SI(NR12)
T(11)=SI(NR10)-SI(NR12)
T(16)=SR(NR15)-SR(NR17)
T(12)=SR(NR11)-SR(NR17)
T(10)=-SI(NR9)-SI(NR16)
T(14)=-SI(NR16)+SI(NR13)
SR(NR2)=T(5)+T(7)
SR6=T(5)-T(7)
SR10=T(6)+T(8)
SR(NR14)=T(6)-T(8)
Q(7)=T(9)+T(10)
Q(8)=T(9)-T(10)
Q(1)=T(11)+T(12)
Q(2)=T(11)-T(12)
Q(4)=T(14)+T(15)
Q(5)=T(15)-T(14)
Q(3)=T(13)+T(16)
Q(6)=T(13)-T(16)
SR(NR1)=Q(3)+Q(7)
SR(NR7)=Q(7)-Q(3)
SR9=Q(8)+Q(6)
SR(NR15)=Q(8)-Q(6)
SR5=Q(1)+Q(4)
SR3=Q(4)-Q(1)
SR13=Q(2)+Q(5)
SR11=Q(5)-Q(2)
SR(NR8)=T(2)
SR(NR4)=T(3)
```

```
      SR12=T(4)
      T(2)=SI(NBASE)-SI(NR1)
      SI(NBASE)=SI(NR1)+SI(NBASE)
      T(4)=SI(NR2)-SR(NR3)
      T(3)=SI(NR2)+SR(NR3)
      T(6)=SI(NR4)-SR(NR5)
      T(5)=SI(NR4)+SR(NR5)
      T(8)=SR(NR6)-SI(NR7)
      T(7)=SR(NR6)+SI(NR7)
      T(9)=SI(NR8)+SI(NR14)
      T(15)=SI(NR8)-SI(NR14)
      T(13)=SR(NR10)+SR(NR12)
      T(11)=SR(NR12)-SR(NR10)
      T(16)=SI(NR15)-SI(NR17)
      T(12)=SI(NR11)-SI(NR17)
      T(10)=SR(NR9)+SR(NR16)
      SI(NR2)=T(5)+T(7)
      SI(NR6)=T(5)-T(7)
      SI(NR10)=T(6)+T(8)
      SI(NR14)=T(6)-T(8)
      Q(7)=T(9)+T(10)
      Q(8)=T(9)-T(10)
      Q(1)=T(11)+T(12)
      Q(2)=T(11)-T(12)
      Q(4)=T(14)+T(15)
      Q(5)=T(15)-T(14)
      Q(3)=T(13)+T(16)
      Q(6)=T(13)-T(16)
      SI(NR1)=Q(3)+Q(7)
      SI(NR7)=Q(7)-Q(3)
      SI(NR9)=Q(8)+Q(6)
      SI(NR15)=Q(8)-Q(6)
      SI(NR5)=Q(1)+Q(4)
      SI(NR3)=Q(4)-Q(1)
      SI(NR13)=Q(2)+Q(5)
      SI(NR11)=Q(5)-Q(2)
      SI(NR8)=T(2)
      SI(NR4)=T(3)
      SI(NR12)=T(4)
      SR(NR3)=SR3
      SR(NR5)=SR5
      SR(NR6)=SR6
      SR(NR9)=SR9
      SR(NR10)=SR10
      SR(NR11)=SR11
      SR(NR12)=SR12
      SR(NR13)=SR13
1620  NBASE=NBASE+16
1630  NBASE=NBASE+NLUP2
1640  NBASE=NBASE+NLUP23
      RETURN
      END
```

APPENDIX D

Prime Factor Algorithm Program

This appendix contains the driver and subroutine for executing and timing the prime factor algorithm (PFA) developed by Burrus (Burrus and Eschenbacher, 1981). The program consists of a single subroutine called using the format

```
CALL PFA(XR,XI,A,B,N,NFT,NI,UNSC,XIN,XOUT)
```

where the variables are defined as follows.

XR -XR is a real array of dimension N. It stores the real component of the input sequence.

XI -XI is a real array of dimension N. It stores the imaginary component of the input sequence.

A -A is a real array of dimension N. It contains the real component output sequence.

B -B is a real array of dimension N. It contains the imaginary component of the output sequence.

N -N is an integer variable specifying the length of the sequence. N must be a product of relatively prime factors from the set 2, 3, 4, 5, 7, 8, 9, and 16.

NFT -NFT is an integer specifying the number of relatively prime factors in N.

NI -NI is an integer array of dimension NFT where each element is a factor of N.

UNSC -UNSC is an integer unscrambling constant used to permute the output sequence into its proper order. UNSC is found by

$$UNSC=(N/n_1)+(N/n_2)+\ldots+(N/n_{NFT})$$

where n_i is a factor of N.

```
C
C *** START OF BOX I, ROUTINE B  ***********
C
C
C*****************************************************************************
C
C  PROGRAM BURRUS PRIME FACTOR ALGORITHM DRIVER
C
C  THIS DRIVER HAS BASICALLY BEEN COPIED FROM THE PRIME FACTOR
C       ALGORITHM DRIVER RECEIVED FROM ROUTE
C
C  AUTHOR:       MARK MEHALIC
C  DATE:         27 APR 83
C  VERSION:      1.0
C
C  CALLING MODULES:     NONE (THIS IS THE MAIN PROGRAM)
C  MODULES CALLED:      BPFA, GRAPH
C
C*****************************************************************************
C
      DIMENSION XR(504),XI(504),A(504),B(504)
      DIMENSION FREQ(504)
      DIMENSION NI(3)
      REAL MAG(504)
      INTEGER UNSC
C
C CHANGE THE VALUE OF T TO CHANGE THE SEQUENCE LENGTH (/100)
C
      N = 504
      T = N/100
      NFT=3
C
C THE FACTORS NI(1), NI(2), NI(3) MUST BE CHANGED FOR DIFFERENT LENGTH
C
      NI(1)=9
      NI(2)=8
      NI(3)=7
      DELT=.01
      UNSC = 191
C
C  DEFINE THE FREQUENCY RECORD LINGTH,DELF:
C
      DELF=1.0/T
C
C  GENERATE THE ARRAY TO BE TRANSFORMED:
C
      PI=3.1415926535898
      E=2.71828
      FO=25.0
      DO 10 I=1,N
      T=(I-1)*DELT
      ARG=2.0*PI*FO*T
      TE=-1.0*T
```

```
      XI(I)=0.
   10 XR(I)=(E**TE)*COS(ARG)
      PRINT*,'TEST'
C
C**********************************************************************
C
C  CALL THE PRIME FACTOR ALGORITHM SUBROUTINE AND DETERMINE TIMING
C
C**********************************************************************
C
      CALL PFA(XR,XI,A,B,N,NFT,NI,UNSC,XIN,XOUT)
      PRINT*,'TEST'
C
C  GENERATE THE REAL, IMAGINARY, AND MAGNITUDE ARRAYS
C
      DO 30 I=1,N
   30 MAG(I)=(( A(I)**2)+( B(I)**2))**0.5
      PRINT*,'TEST'
C
C  GENERATE THE FREQUENCY ARRAY.  AS PER CONVENSION.
C  FREQUENCY TERMS ABOVE THE FOLDING FREQUENCY F/2
C  ARE ASSIGNED TO NEGATIVE FREQUENCIES.
C
C
C**********************************************************************
C
C  PRINT THE MAGNITUDE AND FREQUENCY ARRAYS
C
C**********************************************************************
C
      DO 50 I=1,N
   40 FREQ(I)=((I-1)*DELF)-50.0
      PRINT*,FREQ(I),MAG(I)
50    CONTINUE
      PRINT*,'TEST'
C
C**********************************************************************
C
C  CALL GRAPH TO PLOT THE MAGNITUDE ARRAY
C
C**********************************************************************
C
      CALL GRAPH(MAG,N)
C
C  END OF MAIN PROGRAM
C*******************************************************************
C
C*******************************************************************
      PRINT*,'TEST OVER'
      STOP
      END
C
C  *** END OF BOX I, ROUTINE B  **********
C
```

```
C
C*****************************************************************************
C
C  SUBROUTINE BURRUS PRIME FACTOR ALGORITHM
C
C  DATE:          27 APR 83
C  VERSION:       1.0
C
C  THIS SUBROUTINE WAS TAKEN FROM AN ARTICLE WRITTEN BY C. SIDNEY
C       BURRUS.  IT HAS BEEN MODIFIED TO RUN ON A PDP 11/50 USING
C       THE F4P FORTRAN COMPILER.
C
C*****************************************************************************
C
      SUBROUTINE PFA(X,Y,A,B,N,M,NI,UNSC,XIN,XOUT)
C
C   A PRIME FACTOR FFT PROGRAM
C
C  X,Y   -COMPLEX INPUT DATA ARRAYS. REAL DATA IN X AND
C        IMAGINARY VALUES IN Y.
C  A,B   -COMPLEX OUTPUT VECTORS.  REAL VALUES IN A AND
C        IMAGINARY VALUES IN B.
C  M     -THE NUMBER OF FACTORS OF N
C  N     -SEQUENCE LENGTH WHICH MUST BE FACTORABLE
C        BY MUTUALLY PRIME NOS FROM THE SET (2,3,4,5,7,8,9,16)
C  NI    -ARRAY LENGTH M CONTAINING THE FACTORS OF N.
C  UNSC  -UNSCRAMBLING CONSTANT EQUAL TO N/NI(1)+N/NI(2)+
C        ....+N/NI(M).
C
C   PROGRAM BY C.S. BURRUS
C   RICE UNIVERSITY, AUG 1980
C
      DIMENSION X(N),Y(N),A(N),B(N)
      INTEGER NI(4), I(16), UNSC
      DATA C31, C32 / 0.8660254, 0.5000000 /
      DATA C51, C52 / 0.95105652, 1.5388418 /
      DATA C53,C54 /0.36327126, 0.55901699/
      DATA C55      /-1.25 /
      DATA C71, C72 / -1.16666667, 0.79015647 /
      DATA C73, C74 / 0.055854267, 0.7343022 /
      DATA C75, C76 /0.44095855, 0.34087293 /
      DATA C77, C78 / 0.53396936, 0.87484229 /
      DATA C81      /0.70710678/
      DATA C92, C93 / 0.93969262, -0.17364818 /
      DATA C94, C96 /0.76604444, -0.34202014 /
      DATA C97, C98 / -0.98480775, -0.64278761 /
      DATA C162, C163 / 0.38268343, 1.30656297 /
      DATA C164, C165/ 0.54119610, 0.92387953 /
      XIN=SECNDS(0.0)
      DO 10 K=1,M
      N1=NI(K)
      N2=N/N1
      DO 20 J=1,N,N1
```

```
              I(1)=J
              IT=J
              DO 30 L=2,N1
                   IT=IT+N2
                   IF(IT.GT.N)  IT=IT-N
                   I(L)=IT
30     CONTINUE
                   GO TO(20,102,103,104,105,20,107,108,109,
     C              20,20,20,20,20,20,116),N1
C
C  WFTA N=2
C
102    T1=X(I(1))
       X(I(1))=T1+X(I(2))
       X(I(2))=T1-X(I(2))
       T1=Y(I(1))
       Y(I(1))=T1+Y(I(2))
       Y(I(2))=T1-Y(I(2))
       GO TO 20
C
C  WFTA N=3
C
103    T1=(X(I(2))-X(I(3)))*C31
       U1=(Y(I(2))-Y(I(3)))*C31
       R1=X(I(2))+X(I(3))
       S1=Y(I(2))+Y(I(3))
       T2=X(I(1))-R1*C32
       U2=Y(I(1))-S1*C32
       X(I(1))=X(I(1))+R1
       Y(I(1))=Y(I(1))+S1
       X(I(2))=T2+U1
       X(I(3))=T2-U1
       Y(I(2))=U2-T1
       Y(I(3))=U2+T1
       GO TO 20
C
C  WFTA N=4
C
104    R1=X(I(1))+X(I(3))
       R2=X(I(1))-X(I(3))
       S1=Y(I(1))+Y(I(3))
       S2=Y(I(1))-Y(I(3))
       R3=X(I(2))+X(I(4))
       R4=X(I(2))-X(I(4))
       S3=Y(I(2))+Y(I(4))
       S4=Y(I(2))-Y(I(4))
       X(I(1))=R1+R3
       X(I(3))=R1-R3
       Y(I(1))=S1+S3
       Y(I(3))=S1-S3
       X(I(2))=R2+S4
       X(I(4))=R2-S4
       Y(I(2))=S2-R4
       Y(I(4))=S2+R4
```

```
      GO TO 20
C
C  WFTA N=5
C
105   R1=X(I(2))+X(I(5))
      R2=X(I(2))-X(I(5))
      S1=Y(I(2))+Y(I(5))
      S2=Y(I(2))-Y(I(5))
      R3=X(I(3))+X(I(4))
      R4=X(I(3))-X(I(4))
      S3=Y(I(3))+Y(I(4))
      S4=Y(I(3))-Y(I(4))
      T1=(R2+R4)*C51
      U1=(S2+S4)*C51
      R2=T1-R2*C52
      S2=U1-S2*C52
      R4=T1-R4*C53
      S4=U1-S4*C53
      T1=(R1-R3)*C54
      U1=(S1-S3)*C54
      T2=R1+R3
      U2=S1+S3
      X(I(1))=X(I(1))+T2
      Y(I(1))=Y(I(1))+U2
      T2=X(I(1))+T2*C55
      U2=Y(I(1))+U2*C55
      R1=T2+T1
      R3=T2-T1
      S1=U2+U1
      S3=U2-U1
      X(I(2))=R1+S4
      X(I(5))=R1-S4
      Y(I(2))=S1-R4
      Y(I(5))=S1+R4
      X(I(3))=R3-S2
      X(I(4))=R3+S2
      Y(I(3))=S3+R2
      Y(I(4))=S3-R2
      GO TO 20
C
C
C   WFTA N=7
107   R1=X(I(2))+X(I(7))
      R2=X(I(2))-X(I(7))
      S1=Y(I(2))+Y(I(7))
      S2=Y(I(2))-Y(I(7))
      R3=X(I(3))+X(I(6))
      R4=X(I(3))-X(I(6))
      S3=Y(I(3))+Y(I(6))
      S4=Y(I(3))-Y(I(6))
      R5=X(I(4))+X(I(5))
      R6=X(I(4))-X(I(5))
      S5=Y(I(4))+Y(I(5))
      S6=Y(I(4))-Y(I(5))
```

```
      T1=R1+R3+R5
      U1=S1+S3+S5
      X(I(1))=X(I(1))+T1
      Y(I(1))=Y(I(1))+U1
      T1=X(I(1))+T1*C71
      U1=Y(I(1))+U1*C71
      T2=C72*(R1-R5)
      U2=C72*(S1-S5)
      T3=C73*(R5-R3)
      U3=C73*(S5-S3)
      T4=C74*(R3-R1)
      U4=C74*(S3-S1)
      R1=T1+T2+T3
      R3=T1-T2-T4
      R5=T1-T3+T4
      S1=U1+U2+U3
      S3=U1-U2-U4
      S5=U1-U3+U4
      U1=C75*(S2+S4-S6)
      T1=C75*(R2+R4-R6)
      T2=C76*(R2+R6)
      U2=C76*(S2+S6)
      T3=C77*(R4+R6)
      U3=C77*(S4+S6)
      T4=C78*(R4-R2)
      U4=C78*(S4-S2)
      R2=T1+T2+T3
      R4=T1-T2-T4
      R6=T1-T3+T4
      S2=U1+U2+U3
      S4=U1-U2-U4
      S6=U1-U3+U4
      X(I(2))=R1+S2
      X(I(7))=R1-S2
      Y(I(2))=S1-R2
      Y(I(7))=S1+R2
      X(I(3))=R3+S4
      X(I(6))=R3-S4
      Y(I(3))=S3-R4
      Y(I(6))=S3+R4
      X(I(4))=R5-S6
      X(I(5))=R5+S6
      Y(I(4))=S5+R6
      Y(I(5))=S5-R6
      GO TO 20
C
C  WFTA N=8
C
108   R1=X(I(1))+X(I(5))
      R2=X(I(1))-X(I(5))
      S1=Y(I(1))+Y(I(5))
      S2=Y(I(1))-Y(I(5))
      R3=X(I(2))+X(I(8))
      R4=X(I(2))-X(I(8))
```

```
      S3=Y(I(2))+Y(I(8))
      S4=Y(I(2))-Y(I(8))
      R5=X(I(3))+X(I(7))
      R6=X(I(3))-X(I(7))
      S5=Y(I(3))+Y(I(7))
      S6=Y(I(3))-Y(I(7))
      R7=X(I(4))+X(I(6))
      R8=X(I(4))-X(I(6))
      S7=Y(I(4))+Y(I(6))
      S8=Y(I(4))-Y(I(6))
      T1=R1+R5
      T2=R1-R5
      U1=S1+S5
      U2=S1-S5
      T3=R3+R7
      R3=(R3-R7)*C81
      U3=S3+S7
      S3=(S3-S7)*C81
      T4=R4-R8
      R4=(R4+R8)*C81
      U4=S4-S8
      S4=(S4+S8)*C81
      T5=R2+R3
      T6=R2-R3
      U5=S2+S3
      U6=S2-S3
      T7=R4+R6
      T8=R4-R6
      U7=S4+S6
      U8=S4-S6
      X(I(1))=T1+T3
      X(I(5))=T1-T3
      Y(I(1))=U1+U3
      Y(I(5))=U1-U3
      X(I(2))=T5+U7
      X(I(8))=T5-U7
      Y(I(2))=U5-T7
      Y(I(8))=U5+T7
      X(I(3))=T2+U4
      X(I(7))=T2-U4
      Y(I(3))=U2-T4
      Y(I(7))=U2+T4
      X(I(4))=T6+U8
      X(I(6))=T6-U8
      Y(I(4))=U6-T8
      Y(I(6))=U6+T8
      GO TO 20
C
C  WFTA N=9
C
109   R1=X(I(2))+X(I(9))
      R2=X(I(2))-X(I(9))
      S1=Y(I(2))+Y(I(9))
      S2=Y(I(2))-Y(I(9))
```

```
R3=X(I(3))+X(I(8))
R4=X(I(3))-X(I(8))
S3=Y(I(3))+Y(I(8))
S4=Y(I(3))-Y(I(8))
R5=X(I(4))+X(I(7))
T=-(X(I(4))-X(I(7)))*C31
S5=Y(I(4))+Y(I(7))
U=-(Y(I(4))-Y(I(7)))*C31
R7=X(I(5))+X(I(6))
R8=X(I(5))-X(I(6))
S7=Y(I(5))+Y(I(6))
S8=Y(I(5))-Y(I(6))
R9=X(I(1))+R5
S9=Y(I(1))+S5
T1=X(I(1))-R5*C32
U1=Y(I(1))-S5*C32
T2=(R3-R7)*C92
U2=(S3-S7)*C92
T3=(R1-R7)*C93
U3=(S1-S7)*C93
T4=(R1-R3)*C94
U4=(S1-S3)*C94
R10=R1+R3+R7
S10=S1+S3+S7
R1=T1+T2+T4
R3=T1-T2-T3
R7=T1+T3-T4
S1=U1+U2+U4
S3=U1-U2-U3
S7=U1+U3-U4
X(I(1))=R9+R10
Y(I(1))=S9+S10
R5=R9-R10*C32
S5=S9-S10*C32
R6=-(R2-R4+R8)*C31
S6=-(S2-S4+S8)*C31
T2=(R4+R8)*C96
U2=(S4+S8)*C96
T3=(R2-R8)*C97
U3=(S2-S8)*C97
T4=(R2+R4)*C98
U4=(S2+S4)*C98
R2=T+T2+T4
R4=T-T2-T3
R8=T+T3-T4
S2=U+U2+U4
S4=U-U2-U3
S8=U+U3-U4
X(I(2))=R1-S2
X(I(9))=R1+S2
Y(I(2))=S1+R2
Y(I(9))=S1-R2
X(I(3))=R3+S4
X(I(6))=R3-S4
```

```
      Y(I(3))=S3-R4
      Y(I(8))=S3+R4
      X(I(4))=R5-S6
      X(I(7))=R5+S6
      Y(I(4))=S5+R6
      Y(I(7))=S5-R6
      X(I(5))=R7-S8
      X(I(6))=R7+S8
      Y(I(5))=S7+R8
      Y(I(6))=S7-R8
      GO TO 20
C
C  WFTA N=16
C
116   R1=X(I(1))+X(I(9))
      R2=X(I(1))-X(I(9))
      S1=Y(I(1))+Y(I(9))
      S2=Y(I(1))-Y(I(9))
      R3=X(I(2))+X(I(10))
      R4=X(I(2))-X(I(10))
      S3=Y(I(2))+Y(I(10))
      S4=Y(I(2))-Y(I(10))
      R5=X(I(3))+X(I(11))
      R6=X(I(3))-X(I(11))
      S5=Y(I(3))+Y(I(11))
      S6=Y(I(3))-Y(I(11))
      R7=X(I(4))+X(I(12))
      R8=X(I(4))-X(I(12))
      S7=Y(I(4))+Y(I(12))
      S8=Y(I(4))-Y(I(12))
      R9=X(I(5))+X(I(13))
      R10=X(I(5))-X(I(13))
      S9=Y(I(5))+Y(I(13))
      S10=Y(I(5))-Y(I(13))
      R11=X(I(6))+X(I(14))
      R12=X(I(6))-X(I(14))
      S11=Y(I(6))+Y(I(14))
      S12=Y(I(6))-Y(I(14))
      R13=X(I(7))+X(I(15))
      R14=X(I(7))-X(I(15))
      S13=Y(I(7))+Y(I(15))
      S14=Y(I(7))-Y(I(15))
      R15=X(I(8))+X(I(16))
      R16=X(I(8))-X(I(16))
      S15=Y(I(8))+Y(I(16))
      S16=Y(I(8))-Y(I(16))
      T1=R1+R9
      T2=R1-R9
      U1=S1+S9
      U2=S1-S9
      T3=R3+R11
      T4=R3-R11
      U3=S3+S11
      U4=S3-S11
```

```
T5=R5+R13
T6=R5-R13
U5=S5+S13
U6=S5-S13
T7=R7+R15
T8=R7-R15
U7=S7+S15
U8=S7-S15
T9=C81*(T4+T8)
T10=C81*(T4-T8)
U9=C81*(U4+U8)
U10=C81*(U4-U8)
R1=T1+T5
R3=T1-T5
S1=U1+U5
S3=U1-U5
R5=T3+T7
R7=T3-T7
S5=U3+U7
S7=U3-U7
R9=T2+T10
R11=T2-T10
S9=U2+U10
S11=U2-U10
R13=T6+T9
R15=T6-T9
S13=U6+U9
S15=U6-U9
T1=R4+R16
T2=R4-R16
U1=S4+S16
U2=S4-S16
T3=C81*(R6+R14)
T4=C81*(R6-R14)
U3=C81*(S6+S14)
U4=C81*(S6-S14)
T5=R8+R12
T6=R8-R12
U5=S8+S12
U6=S8-S12
T7=C162*(T2-T6)
T8=C163*T2-T7
T9=C164*T6-T7
T10=R2+T4
T11=R2-T4
R2=T10+T8
R4=T10-T8
R6=T11+T9
R8=T11-T9
U7=C162*(U2-U6)
U8=C163*U2-U7
U9=C164*U6-U7
U10=S2+U4
U11=S2-U4
```

```
S2=U10+U8
S4=U10-U8
S6=U11+U9
S8=U11-U9
T7=C165*(T1+T5)
T8=T7-C164*T1
T9=T7-C163*T5
T10=R10+T3
T11=R10-T3
R10=T10+T8
R12=T10-T8
R14=T11+T9
R16=T11-T9
U7=C165*(U1+U5)
U8=U7-C164*U1
U9=U7-C163*U5
U10=S10+U3
U11=S10-U3
S10=U10+U8
S12=U10-U8
S14=U11+U9
S16=U11-U9
X(I(1))=R1+R5
X(I(9))=R1-R5
Y(I(1))=S1+S5
Y(I(9))=S1-S5
X(I(2))=R2+S10
X(I(16))=R2-S10
Y(I(2))=S2-R10
Y(I(16))=S2+R10
X(I(3))=R9+S13
X(I(15))=R9-S13
Y(I(3))=S9-R13
Y(I(15))=S9+R13
X(I(4))=R8-S16
X(I(14))=R8+S16
Y(I(4))=S8+R16
Y(I(14))=S8-R16
X(I(5))=R3+S7
X(I(13))=R3-S7
Y(I(5))=S3-R7
Y(I(13))=S3+R7
X(I(6))=R6+S14
X(I(12))=R6-S14
Y(I(6))=S6-R14
Y(I(12))=S6+R14
X(I(7))=R11-S15
X(I(11))=R11+S15
Y(I(7))=S11+R15
Y(I(11))=S11-R15
X(I(8))=R4-S12
X(I(10))=R4+S12
Y(I(8))=S4+R12
Y(I(10))=S4-R12
```

```
      GO TO 20
20      CONTINUE
10      CONTINUE
C
C  UNSCRAMBLING
C
      L=1
      DO 2 K=1,N
      A(K)=X(L)
      B(K)=Y(L)
      L=L+UNSC
      IF(L.GT.N) L=L-N
2     CONTINUE
      XOUT=SECNDS(XIN)
      PRINT*,'THE PFA EXECUTION TIME IS  ',XOUT,'SECONDS'
      RETURN
      END
```

Vita

Mark Andrew Mehalic was born on 11 May 1958 in Latrobe, Pennsylvania. In 1976, he graduated from Derry Area Senior High School in Derry, Pennsylvania. He attended the Pennsylvania State University from which he received a Bachelor of Science in Electrical Engineering with high distinction in May 1980. Upon graduation, he was designated an AFROTC distinguished graduate and received a commission in the United States Air Force. He entered active duty 25 September 1980 and was assigned to the Flight Dynamics Laboratory, Air Force Wright Aeronautical Laboratories, Wright-Patterson Air Force Base, Ohio. On 2 June 1982 he was assigned to the School of Engineering, Air Force Institute of Technology.

Permanent Address: R. D. #1 Box 199
Derry, Pennsylvania 15627

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GE/EE/83D-47	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) See Box 19	

12. PERSONAL AUTHOR(S)
Mark A. Mehalic, First Lieutenant, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
MS Thesis	FROM ______ TO ______	1983 December	286

16. SUPPLEMENTARY NOTATION

Approved for public release: IAW AFR 190-17.
7 Feb 84
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFC)
Wright-Patterson AFB OH 45433

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Fast Fourier Transform, Computer Architecture, Discrete Fourier Transform, Computer Performance
09	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Title: EFFECTS OF COMPUTER ARCHITECTURE ON FFT ALGORITHM PERFORMANCE

Thesis Advisor: Pedro L. Rustan, Captain, USAF

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Pedro L. Rustan, Captain, USAF	513-257-7469	AFWAL/FIES

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

This study examines the effects of computer architecture on FFT algorithm performance. The computer architectures evaluated are those of the Cray-1, CDC Cyber 750, IBM 370/155, DEC VAX 11/780, DEC PDP 11/60, DEC PDP 11/50, and Cromemco Z-2D. The algorithms executed are the radix-2, mixed-radix FFT (MFFT), Winograd Fourier Transform Algorithm (WFTA), and prime factor algorithm (PFA).

The execution time of each algorithm for different sequence lengths is determined for each computer. The initialized WFTA is fastest on the Cray-1, the radix-2 is fastest on the CDC Cyber 750, and the PFA is fastest on the others. Then the number of assembly language instructions executed are determined for the following categories: data transfers, floating point additions and subtractions, floating point multiplications and divisions, and integer operations. The correlation coefficients between the number of assembly language instructions in each category and the algorithm execution speeds are determined for each computer. The average values for the correlation coefficients range from 0.8614 for the floating multiplications and divisions to 0.9792 for the data transfers. The values of the correlation coefficients are then related to the computer architectures.

The computer architectures are then compared against each other to determine what features are desirable in an FFT processor. The most desirable features are assembled into a proposed minimum computer architecture for efficient FFT performance. The minimum architecture includes separate functional units, a cache memory, and separate floating point and integer registers. In addition, a method for deciding how to improve a given architecture is presented. The method is based on the correlation coefficients for that architecture. Guidelines for predicting FFT algorithm performance are given based on the known computer architecture. Floating point processors execute the radix-2 fastest, data transfer processors execute the PFA fastest, and vector processors execute the initialized WFTA fastest.

DAT

FILM